TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MUHAMMAD HUSHAM YOUSAF

ANALYSIS OF SIMULATOR FEASIBILITY IN DEVELOPMENT OF WIRELESS SENSOR NETWORK APPLICATIONS

Master of Science Thesis

Examiner: Prof. Jarmo Harju
Examiner: Dr. Teemu Laukkarinen
Examiners and topic approved on 29 March 2017

ABSTRACT

MUHAMMAD HUSHAM YOUSAF: Analysis of simulator feasibility in development of wireless sensor network applications
Tampere University of technology
Master of Science Thesis, 52 pages
June 2017
Masters Degree Programme in Information Technology
Major: Communication Systems and Networks
Examiners: Professor Jarmo Harju and Postdoctoral Researcher Teemu Laukkarinen

Keywords: Wireless Sensor Network (WSN), simulator, feasibility analysis, simulator comparison, COOJA

Wireless Sensor Networks (WSNs) consist of small autonomous electronic devices that sense environmental conditions and communicate over wireless links. WSNs have applications in various fields of science and technology, which are diverse and typically exist in remote areas with harsh conditions. This makes testing and debugging of applications time consuming, costly and sometimes even impossible for application developers. Hence, simulators are used to develop and test WSN applications to verify the functionality of applications without actual deployment and to avoid debugging and testing costs.

Selection of a feasible simulator for WSN application development requires analysis and comparison of available WSN simulators. A feasible simulator fulfils application requirements and presents related features. Existing comparison surveys and articles present strengths and weaknesses of WSN simulators. They do not concentrate on the attributes of simulators, which an application developer should consider in selecting a feasible WSN simulator before application development.

In this thesis, WSN simulators are analyzed and their attributes are collected, which can be used in selecting a feasible simulator for application development. Three types of attributes are collected: 1) activity attributes, 2) basic attributes, 3) core attributes. Activity attributes present how active and up-to-date the WSN simulators are. Basic attributes specify the type, category and development language of WSN simulators. Core attributes present the availability of core WSN requirements such as, scalability, consumed power and memory calculation modules in simulators. Seven simulators are compared using these attributes. Requirements for typical WSN applications are defined and consist of multi-hop routing, calculation of execution time, power and memory consumption of nodes. The feasibilities of the simulators are measured against the application requirements. COOJA meets all the requirements and seems feasible for WSN application development and testing.

Feasibility of COOJA simulator is verified experimentally by developing three test applications. In the first application, COOJA simulates multi-hop network and provides four types of statistics, which are network, power, sensor and topology statistics. In the second application, COOJA measures power and memory consumption. In the third application, power consumption is measured for each function call of the application program. COOJA fulfils all requirements and is found feasible for WSN application development and testing.

PREFACE

This thesis work was carried out for the Department of Pervasive Computing, Tampere University of Technology, Finland during the year 2015. The thesis work was paid and I am very thankful to my supervisor, Teemu Laukkarinen, for providing the work opportunity and the financial support.

The guidance provided by Teemu was very clear, complete and valuable specially in thesis writing process. I am also grateful for his patience and kindness during the course of thesis. It was a great learning experience, and I will surely reuse the attained knowledge in the rest of my life.

I am also very grateful to my thesis examiner, Jarmo Harju. His comments and suggestions were valuable and helped in improving the presentation of work.

I also want to thank Almighty God for His countless blessings, which kept me motivated. I also want to express my gratitude to my parents for their prayers and efforts, which they did in order to see their son educated.

In the end, I would like to appreciate my beloved wife, Murva Moazzam, for her love, patience the food she cooks and taking care of our cute little baby girl, Aylin.

Tampere, 31.07.2017

Muhammad Husham Yousaf

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	Artificial neural network
CPU	Central processing unit
ELF	Executable and linkable format
ESB	Embedded sensor board
GUI	Graphical user interface
HAL	Hardware abstraction layer
ID	Identification
IP	Internet protocol
I/O	Input/Output
JNI	Java native interface
LED	Light emitting diode
LQI	Link quality indicator
MAC	Medium access control
OS	Operating system
RAM	Random access memory
ROM	Read only memory
RPL	Routing protocol for low power and lossy networks
RSSI	Received signal strength indicator
TCP	Transmission control protocol
UDP	User datagram protocol
WSN	Wireless sensor network

1. INTRODUCTION

Advancements in the field of computing technology have led in the development of high-speed computing systems that model and solve real-life problems, which were time consuming and costly to resolve in the past. Simulators are widely used in scientific research and development to model real-life objects and provide an artificial environment where modelled objects exist. These objects can be anything that have quantifiable properties and characteristics in real-life. The objects are then tested, debugged and observed under simulated environment rather than in real-life environment to get knowledge of their working in quick and inexpensive way.

Simulators have applications in many fields of science and technology. In the field of digital communication, communication devices and physical medium are simulated to evaluate and test the communication process in varying conditions. The reliability of simulator outcomes depends on how precisely the simulator can mimic real-life objects [1]. The availability of models and other supportive simulator functions, such as wireless propagation models, enable and make the simulators feasible for research and development.

1.1 Wireless sensor networks

Wireless sensor network (WSN) consists of small autonomous electronic devices communicating with each other via radio interface. The sensing devices called nodes have the sensing module, processing unit, memory, power source and radio module. WSN nodes are tiny and are limited in power, processing and radio resources. They have their own embedded operating systems and communication protocols as the traditional communication protocols are too resource consuming for the limited resources [2].

WSN applications are continuously developed in variety of fields ranging from environmental monitoring and factory automation to health care and military surveillance [3]. Typically, a WSN is installed over a geographical area to monitor the physical phenomenon, such as temperature or light intensity. The number of nodes and size of geographic area depend upon the application of WSN. If an application requires WSN deployment over a city area, then thousands of nodes would be needed, whereas, tens of nodes would be enough for an office building surveillance application.

WSN application requires rigorous testing and debugging of the communication protocols, application software, and other supportive software before the actual deployment,

because the nodes may be hard or even impossible to access afterward. A software mistake or unforeseen situation may destroy the whole deployment [4]. Therefore, the expected functionality must be verified to a high degree before the deployment. WSN simulators that contain the testing capabilities of application software and protocols can be used for the verification. It will make WSN application development quicker and inexpensive by allowing the verification without actual deployment [5] .

1.2 WSN simulators

Number of WSN simulators have been developed, such as TOSSIM [6], NS-2 [7], COOJA [8] that support WSN application development, testing and debugging. Because of varying needs of WSN applications, every simulator targets different aspects of WSN and offers its own set of features. For example, NS-2 is a pure packet level simulator. It simulates network packets and keeps track of packets sent or received among the nodes. On the other hand, it does not calculate the power consumption of node. Hence, it is not feasible to select NS-2 for WSN application development where power consumption of node is needed.

Selection of a WSN simulator requires analysis of available WSN simulators and application requirements. The design goals, strengths and weaknesses of simulators must be known and compatible with the application under development. Articles and comparative surveys of WSN simulators assist in the selection process of feasible simulator. The articles [8] [10] [11] provide comparison of features and limitations of available WSN simulators. These articles focus mainly on providing the description of simulators abilities. They do not concentrate on the simulator attributes, which an application developer should consider before selecting a feasible WSN simulator.

1.3 Scope and methods

This thesis analyzes the feasibly of WSN simulators for application development. The thesis presents the comprehensive tables of simulator attributes, which a WSN application developer can use for selecting a feasible WSN simulator for application development. Seven WSN simulators are compared using these attribute tables. The best fit simulator, COOJA, is further analyzed and tested experimentally to see that how it fulfils the basic requirements of WSN application development to verify its feasibility. The feasibility analysis consists of setting the requirements for three different WSN test applications, developing the applications, and comparing the results to the requirements.

The research methods used in this thesis consist of literature review of WSN simulators and the exploratory study of COOJA. Initially, the comparison surveys of WSN simulators, research papers, thesis reports and simulators websites are reviewed to know the existing knowledge about simulators and to find the knowledge gaps. Then, COOJA simulator is explored in detail by reviewing its technical documentation before application

development. In the end, WSN applications are developed to verify the feasibility of COOJA.

1.4 Thesis structure

This thesis is divided into two parts.

The first part consists of Chapter 2 and Chapter 3, which present the properties of WSNs, WSN applications and WSN simulators. Chapter 2 presents WSN components, WSN applications and WSN verification methods as necessary information. Chapter 3 explains WSN simulation models and classify WSN simulators. Further, this chapter collects the simulator attributes, compares seven WSN simulators and analyzes their feasibilities for application development.

The second part consists of Chapter 4 and Chapter 5, which covers WSN application development using COOJA. Chapter 4 explains core components and characteristics of COOJA and Contiki operating system. In Chapter 5, three test applications are developed and feasibility of COOJA is analyzed.

Finally, Chapter 6 concludes the thesis.

2. WSN PROPERTIES

This chapter explains basic WSN concepts needed to understand the thesis, such as WSN infrastructure, WSN node, applications and verification methods.

2.1 WSN infrastructure

A WSN is defined as a network of multiple electronic devices that communicate with each other over a wireless medium [12]. These devices, called *nodes* are tiny electronic devices and perform different tasks, for example, sensing of natural phenomena, collection or relaying of data in the network. In a typical WSN application, *sensor nodes* produce data by sensing the environment with sensors, such as temperature, humidity, vibration, or motion detector sensors. This data is then forwarded to *sink node* either directly or through intermediate relay nodes. Sink nodes also serve as gateways to WSN backbone infrastructure and to other networks, such as the Internet as shown in Figure 2.1 [13]. The backbone infrastructure administers nodes and networks and typically consists of databases for storing data, user interfaces to interact with the network and data analysis tools for data visualization and decision making.

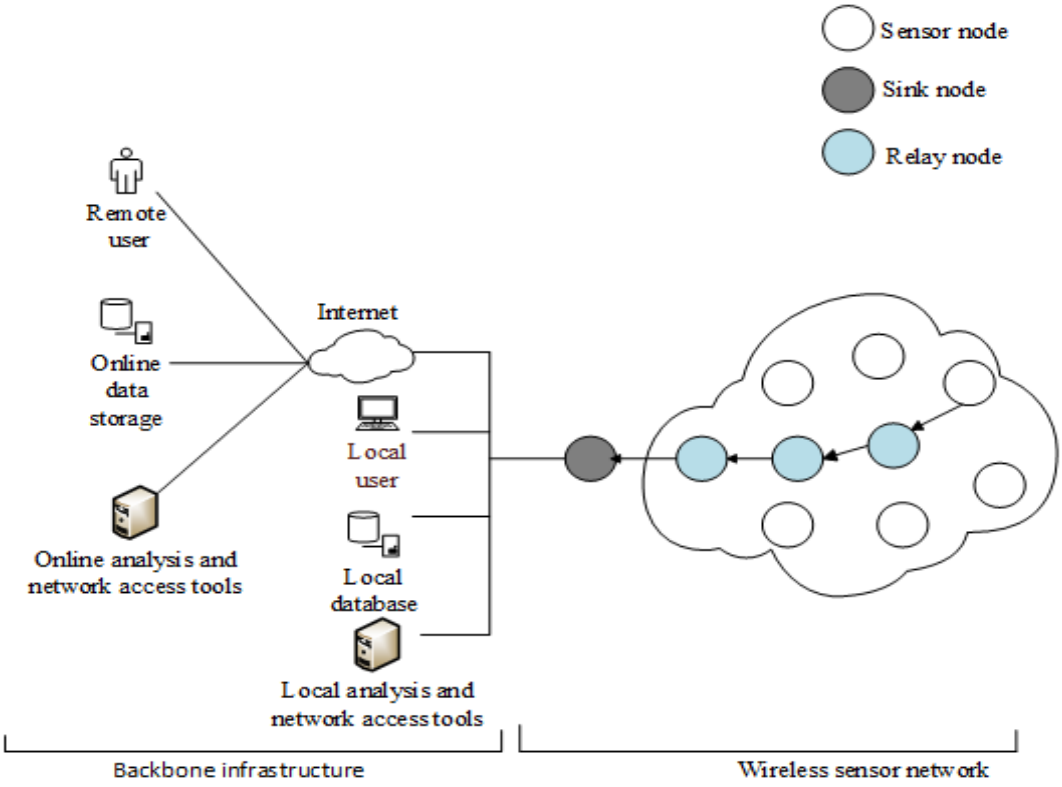


Figure 2.1 *WSN infrastructure*

2.2 WSN platform – WSN node

WSN applications are diverse with varying requirements [12] . For example, an environmental monitoring application requires periodic measurements with low energy, radio, and memory resource consumption, whereas, an object tracking application of troops requires real-time measurements of motion sensors with higher energy, radio, and memory resource consumption. Therefore, the nodes are specifically designed to meet the requirements of WSN applications.

A typical WSN node consists of four basic hardware components [14]. The components are shown in Figure 2.1 and described below:

1. *Sensing unit* monitors the environment and measures the physical phenomena, for example, temperature, vibration etc.
2. *Computing unit* consists of subcomponents like microcontroller, data storage, such as, read only memory (ROM), random access memory (RAM), or flash. Furthermore, it also includes I/O ports, timers and other supporting peripherals, such as, analogue to digital converter.
3. *Radio unit* allows the exchange of information via wireless transceivers and antenna.
4. *Power unit* provides system supply voltage to all other units of node. Power unit may get energy from batteries, energy scavenging or by wired power sources.

WSN node runs the software that implements the WSN and consists of the components shown in Figure 2.2 [15]. A node may use an Operating System (OS) to manage the hardware components through a set of code routines called hardware abstraction layer (HAL) [16]. HAL provides application developers access to underlying hardware through a programming interface. In addition, OS implements network protocol stack that allows nodes to form the WSN and communicate with each other. WSN OSs are designed in such a way that they can fit in the limited memory of node and consume the scarce resources in an efficient manner [17] .

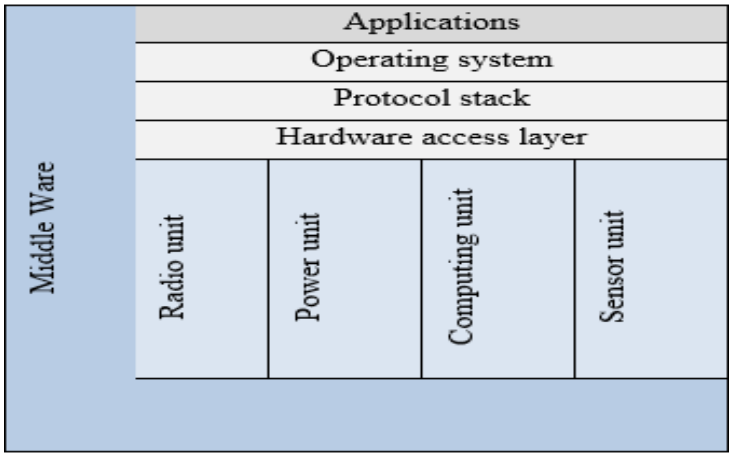


Figure 2.2 *Components of WSN node*

2.3 Protocol stack and middleware

WSN network protocols are typically presented in layered architecture called protocol stack [18] [20].

Physical Layer: Physical layer implements the radio communication hardware. Exchange of bits takes place over a radio link between the nodes. Main functions include: transmit power adjustment, selection of frequency channel, modulation and demodulation, analogue to digital conversions and vice versa. In addition, physical layer may also measure received signal strength indicator (RSSI) and link quality indicator (LQI), cyclic redundancy check for checking the errors and encrypt and decrypt the messages for network security.

Medium Access Control (MAC): MAC establishes the link between the nodes after discovering the neighbor and then exchanges frames with neighbors. MAC layer is responsible for network access functions, such as, radio channel access, transmission error control, frame queue control and assembly of frames. Furthermore, it may perform management functions, such as, network scanning, sleep cycles and distance estimations.

Network Layer: Network layer helps a node in deciding to which node it should send the message. The major functions include: multi-hop routing, route discovery, route construction, next hop selection, packet assembly, packet queue control, data forwarding, and route maintenance. Internet Protocol version 6 (IPv6) is an example of network layer protocol.

Transport Layer: Transport layer provide functions, such as end to end reliable delivery of messages and congestion control in wired and wireless networks. Typical transport layer protocols, such as Transmission Control Protocol (TCP) and User Datagram Proto-

col (UDP) are complex and resource consuming for WSNs and are used after modification. For example, Contiki OS uses a simplified version of UDP protocol called simple-udp for message delivery.

Application Layer: Application layer runs the applications programs, which typically consists of the algorithms and procedures to perform specialized tasks. These tasks are defined by the application requirements and usually are request-response interaction, event notifications, data collection, data aggregation or data fusion. Also, these application programs take decisions, for example, at what time or under which conditions the data should be collected from the senor, or at what time or under which conditions data should be sent to sink node.

Middleware: Middleware combines all the services of protocols stack, operating system, HAL and present it to application layer to access resources of the node directly.

2.4 WSN topology

WSN topology refers to the placement of WSN nodes and how they communicate in a network [20]. Communication in a WSN is single-hop or multi-hop or both depending upon the size and configuration of the network at specific instance of time [21]. In single-hop communication, two nodes communicate directly because they lie within the communication range of each other. While, in multi-hop communication, two communicating nodes do not lie within the communication range of each other and hence need relay nodes for relaying their data. Communication in the network can be both uni-directional and bi-directional. However, the communication remains uni-directional most of the time because, in WSN, data usually travels from sensor nodes to sink node [22].

2.5 WSN essentials

Although, every scenario of WSN has its own requirements and needs, but the following characteristics should be ensured to meet the basic needs of most of the applications. [23] - [26]

Scalable: WSN should be scalable. The network should support addition of new nodes without disrupting the network.

Real-time behavior: The desired data from the network should be available in a certain guaranteed time without long time delays. For example, in alarm applications, events occurred in environments should be reported quickly without delays.

Available and identifiable: Network resources, for example, access to nodes should remain available without long breaks. Nodes should also be easily identifiable and locatable in the network.

Fault tolerant, autonomous and self-configurable: WSNs are dynamic in nature, new nodes come and network topology often changes. Also, the interference from weather, for example, rain or storm may cause connectivity problems among nodes in the network. These dynamics demand the network to act autonomously, recover itself and remain operational. This is achieved, for example, by finding alternative communication paths between the communicating nodes.

Efficient design and long-life: WSN hardware, algorithms and protocols optimization should be considered for energy conservation in advance. WSN applications should also be carefully developed so that they only consume the amount of energy, which is necessary to perform the desired operations. Low power designs use the scarce energy resource efficiently and keep the network operational for longer period, thus, increasing the life of the network.

Low-cost: WSN nodes are small and required in abundance. They are also deployed in tough areas, for example, around a volcano or inside a power turbine. Change of batteries or maintenance of a node could be costly and difficult. Sensor nodes should be low-cost so that they can be replaced or even disposed if needed.

Secure: The communication in the network must be secure and should be only available to the users of network. WSN may hold private and confidential data, for example, in military applications, where data about movement of troops may be gathered. This data should not be stolen in any case because of it critical nature.

2.6 WSN applications

Wireless sensor network applications exist in many fields. Environmental monitoring, agriculture, military surveillance and healthcare are few examples. WSN applications can be classified into following application areas:

Environment Monitoring applications: Environment monitoring applications are deployed in open environments and monitor the large areas, for example, a lake or a glacier [27]. Most common physical phenomenon that are monitored by WSNs are temperature, pressure, vibration, humidity, intensity of light, radiation, chemical composition and location. Applications also require large number of nodes to cover the area under observation. For example, the number of nodes in covering forest area could reach hundreds.

Military applications: As WSNs are autonomous, self-configurable, and deployable in remote areas, they are feasible for various military applications [28]. These include applications, such as, a border security application, which track the movement of troops or immigrants, or an ad hoc WSN application for mobile tanks and troops, which share the critical information regarding battle tactics and position of enemy on battle field in real-

time. Military applications usually demand secure network communication, fast deployment of network and communication range around 250 - 500m or more in some cases. Furthermore, network should be able to tolerate network jamming and interception by enemies.

Industrial applications: Sensors are used to monitor and control industrial processes [29]. For example, in ceramics industry, air pressure controls airflow within the manufacturing kiln, which can be measured and adjusted through sensor networks. In industrial automation, WSNs are usually deployed indoors. A wireless sensor can be behind concreate wall or it can also be inside a metal kiln. A wireless network should be designed after considering the hostile environment in which a network has to exist, because a node transmits through a radio medium. Materials, for example, concrete affects the propagation of radio waves. Also, the interference produced by machines and plants degrades the signal quality. Other requirements include, autonomous operations of network without human intervention, low maintenance and repairing costs, data confidentiality and reliability.

Surveillance and Alarm applications: WSNs are handy in applications that aim to provide security [27]. WSNs can be deployed inside an office building, airports or private property to provide security. Such applications use motion sensors to detect the presence of objects like human or vehicle inside an application territory. An alarm is called when some ambiguity or irregularity is detected. Such applications require data confidentiality, network reliability and stability.

City automation: WSNs are integral part of smart city applications [30]. Smart city applications require data for analysis and decision making about city functions in order to manage city resources efficiently and to improve the quality of life. For example, how much power need to be generated during weekends or during weekdays, which car lanes remain less crowded and can be used during traffic peak hours are some of decisions, which smart city applications take. The data is provided by the wireless sensor networks deployed to monitor city resources. The data is aggregated from sensor networks, stored in data servers and then analyzed before decision making. Grid stations, dams, lakes, streets, train stations, airports are some of the examples of resources managed in smart city applications.

Healthcare Applications: WSNs are useful for healthcare applications for example, collection of patient health data at regular interval of time, tracking movement of the patients inside hospital or reminding patients about their medication and meals [31]. The patients are attached with wireless sensors, such as, a wearable wireless sensor, which measures heart rate, blood pressure, temperature at regular intervals or in real-time. These measurements can be collected periodically or in real-time depending upon the condition of patient. Also, the measurements can be transmitted periodically or in real-time to hospital staff, which includes doctors, nurses or other supporting staff. An emergency could be

declared if measurements, which are irregular and critical in nature are received. For example, an irregular heartbeat is reported immediately to concern staff and a life can be saved by prompt handling of patient. WSNs are effective in such situations where critical data is quickly disseminated without human intervention and results in saving human lives.

2.7 WSN application functions

After observing the WSN infrastructure and WSN applications, it is found that WSN tasks are performed by the cooperation of nodes in the network. These nodes share the burden of overall application logic distributed by the application developers. The hardware components and software of nodes are planned, developed and configured in such a way that they fulfil the objectives of WSN application collectively and in an efficient manner. The functions of WSN applications, which an application developer need to address during WSN application development are given below:

Application type and data flow models: The type of application helps application developer in deciding flow of data in network. Data flow from sensor nodes to sink nodes can be periodic, real time or event triggered.

Routing: After deciding data flow models, routing in the WSN defines how data will be transferred from source to destination. Routing can be single-hop or multiple-hop. There are number of routing protocols available for WSN application development, for example, direct diffusion protocols, energy-aware cluster based protocols.

Area coverage, topology and node requirements: Area coverage, application type and network topology helps in deciding the type, number and hardware configuration of nodes to be used in an application.

Sensor node program: Application developers develop application programs that run on sensor nodes. These programs sense the environment and send the data to sink nodes by using routing protocols. Application developer must consider network formation and maintenance functions, such as, assignment of IP addresses, construction of routing paths by using services offered by operating system of node.

Processing at sink node: Application programs are also developed for sink nodes. These programs are used to collect and store the data from multiple sensor nodes. Also, these programs allow backbone infrastructure to communicate with the nodes present in network.

Processing at relay node: Application programs for relay nodes are typically interested in forwarding the received data towards destination by using routing protocols. These programs can also process the data, for example, average of the received temperature measurements can be calculated and then forwarded towards the destination.

Processing at backbone infrastructure: Application programs for backbone infrastructure involve in storage, presentation, analysis and evaluation of data received from sensor network.

2.8 Verification methods of WSN functionality

Verification of WSN functions is undertaken either through test-beds or simulations before actual deployment [32]. Verification after deployment of WSN in remote places is limited because it is costly and time consuming and sometime even impossible due to poor operational conditions. Traditionally, the main techniques for verifying the functions of wired and wireless networks are analytical methods, computer simulation, and test-beds.

Analytical methods in sensor networks are not effective enough because of large number of constraints, such as, scarce resources, decentralized collaboration and self-configuration.

Test-beds are valuable but implementing the testbeds on large scale is difficult. Test-beds are costly and time consuming to set up and difficult to maintain. They are typically setup in small scale and provide limited topology, hardware and software configurations.

Simulations have limitations as well but simulations are considered an important phase during protocol design, development and testing because simulations provide easier, faster and cheaper way to evaluate protocols and algorithms [33]. Simulation model do not provide exact hostile environment in which a WSN has to operate, however it allows the testing of various WSN techniques, such as radio propagation, link quality, medium interferences and data flow in topology [34]. Also, it may provide graphical user interface to have more control over the nodes. Thus, it seems that simulation is the most reasonable approach to evaluate the WSN at low cost and in short time.

3. WSN SIMULATORS

Simulation plays a key role in developing quick and low cost applications. However, not all simulations produce satisfactory results. Correctness of simulated model and its appropriateness for the application development must be considered before performing simulations [31]. This chapter defines and explains key concepts related to WSN simulators, such as simulation model, correctness of simulation model, types of simulations and attributes of simulators.

3.1 Model, simulation model, simulator, simulation

A *model* is simplified representation of an object, system or subsystems. These representations are typically in mathematical or visual form. Models are also represented in the form of a computer program, such as, a *simulation model*. In a simulation model, a system or sub system is modelled or simulated and studied on computers by using the specifically designed software programs called *simulators*, such as COOJA, which is a WSN simulator. In a WSN simulation, a specific application of WSN is defined and network of sensor nodes and wireless medium are modelled and observed to gain the knowledge of their working in varying operational conditions, for example, working of WSN network consisting of ten sensor nodes and one sink node at different radio noise levels.

3.2 Correctness of simulation model

A simulated model is simplified form of real-life object or a system. It is also limited in scope because simulated model only includes modeling of events and factors, which are assumed to be relevant to the working of application of a system or an object. Simulation model does not model every aspect of real-life objects. Unlimited number of uncontrolled events may occur, which may or may not be relevant to the working of real-life objects, for example, an earth quake, flood, or noise from unidentified object in the surroundings of the object under observation. If the model is over simplified then simulation results become useless as many factors, which were relevant, might be ignored [1]. Therefore, the model has to be based on solid assumptions in order to get trustful results. However, if the model tends to move closer to reality by increasing number of simulated factors then the complexity and computational demands of simulator increase. The tradeoff of simulation always remain in between the correctness and performance of the model.

The correctness of simulated model depends upon following assumption [33] [36]:

Model accuracy: Simulation model is accurately implemented by developers. This means that representation of real-life objects lie within acceptable range and the model

has provided sufficient number of sub models, which are relevant for application development. Model has also implemented the desired functionality correctly.

Model verification and validity: Model accuracy is verified by experts of real-life systems by analyzing the results produced by the simulation model. For example, input-output validity tests help the experts in deciding the correctness of a model. In these tests, the same set of inputs is given to both simulated model and real-life systems. Then, outputs from both are obtained and compared to evaluate the correctness of simulated model.

Data availability and integrity: It is assumed that sufficient amount of the data about real life objects is available. Also, the model is built after studying the relevant data from reliable resources.

3.3 WSN simulation model

A basic WSN simulation model simulates the following WSN components [10]:

WSN node: Emulators are embedded in simulation models to copy the behavior of hardware components of node. For example, COOJA uses MSPsim [37] and Avora [38] emulators to emulates MSP430 [39] and AVR [40] microprocessors respectively. Simulation models also include the operating system components required to manage the functions of node in a simulation.

Wireless Medium: Radio propagation models are included in WSN simulation models. These models define radio channel properties, for example, channel bandwidth. Also, such models explain how the signal will travel between sender and receiver and how the signal parameters, such as, signal to noise ratio, will affect the signal during its propagation.

Environment: WSN nodes exist and communicate in an environment. Environment model carries all those events that stimulates the network, for example, a vibration, rise in temperature, a “button click” on node or a “time tick”.

Event generator: This includes modeling of entities that generate the events in the environment. For example, modeling of a device that generates noise or heat in the environment or a timer.

Mobility: Mobility models represent movement of the nodes in the network and define how velocity, acceleration and location of nodes will change in the network.

Furthermore, the simulation model provides a development and debug environment to view, edit, run and debug WSN applications as shown in Figure 3.1. The model also presents output data in user-friendly forms, such as graphs and tables.

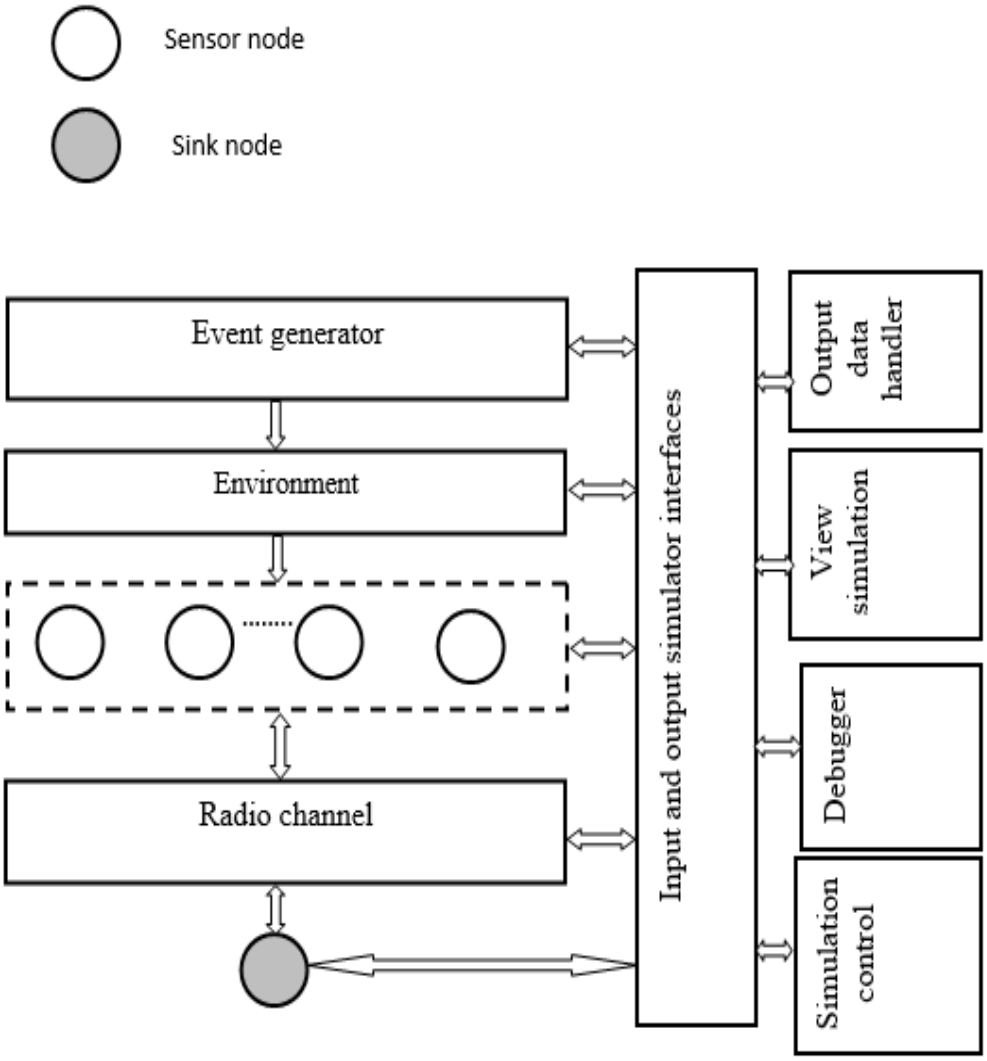


Figure 3.1 *Components of simulation model*

3.4 Types of WSN simulations

Simulations typically run in synchronous mode or in event triggered mode [10]. Events in a simulation happen randomly in the fixed time slots and are called discrete events.

Synchronous simulation: Synchronous simulation works on loops. Simulator increments global time by one unit at the beginning of each loop. Then, it iterates over each node one by one and updates the node position and other node parameters, such as power consumption according to the mobility and battery models of node.

Discrete event simulation: These simulations are event based. The simulator holds a queue of message events and timer events. The simulator repeatedly picks the most recent event and processes it. These simulations are usually faster than the synchronous simulations because in synchronous simulation, simulator loops over all nodes one by one and

performs fixed steps even if the nodes are sleeping. In discrete event simulations, only message and timer events from the queue are processed.

3.5 WSN simulation definition

WSN simulation is defined by configuring the WSN components, which are required in the development of WSN applications.

The process of defining a simulation includes the following decisions:

- The number and the type of nodes used in a simulation. For example, twenty temperature sensing nodes and two sink nodes in a network.
- The propagation models used, for example, ground wave models, point-to-point propagation models, indoor or outdoor propagation models.
- The placement of nodes and the flow of data in the network.
- The mobility of nodes, for example, configuration of velocity, acceleration and location of nodes.
- The operating system used for nodes, for example, Contiki [8], or TinyOS [41]
- The type and number of application software used on each node.
- The type and number of entities generating events and noise in the environment.

3.6 Running and debugging a simulation

WSN simulations are run and activity of the WSN network is monitored in activity viewer windows or tools provided in simulators. The simulations can be paused, stopped and re-run with new configurations. Debug environment provide tools to hold software execution and allow viewing and setting of variable values at run-time.

3.7 Classification of simulators

WSN simulators are designed with specific design goals. Some simulate hardware components of nodes, whereas, others capture operating system behavior. WSN simulators targeting different aspects of WSN are shown in Figure 3.2.

WSN simulators are broadly classified into following categories [42][43]:

Network level or packet level simulators: These simulators focus on the networking aspect of WSN. They provide detailed statistics about the networking protocols, such as, routing protocols. They collect information, such as number of packets sent and received in the network, size and content of packets, propagation delay and number of hops. However, these simulators provide less details about hardware components. Ns-2 is an example of network level simulators [7].

Algorithm and protocol level simulators: Algorithm level simulators focus on optimization of WSN algorithms and data structures, for example, AlgoSensim [44] analyzes distributed routing algorithms in WSNs.

Hardware level/instruction level simulators or emulators: These simulators replicate the behavior of hardware components of sensor node. For example, MSPsim is an instruction level emulator, which captures the behavior of microprocessor execution. It records values of each register during code execution. Emulators are helpful in computing the precise power consumption of node hardware. They also help in developing and rectifying the device drivers.

Operating System level simulators: These simulators focus on simulating operating systems. For example, COOJA simulate Contiki and TOSSIM simulates TinyOS. These simulators are used to debug and develop protocols and algorithms, such as, protocol stack or memory allocation algorithms.

Cross level Simulators: These simulators simulate nodes at multiple levels of abstraction. For example, COOJA simulates at both hardware and operating system levels. Cross level simulators capture more detailed simulations than simulators that only capture details at single level. They are useful in heterogeneous networks where network consists of multiple types of nodes and abstraction at multiple level is needed.

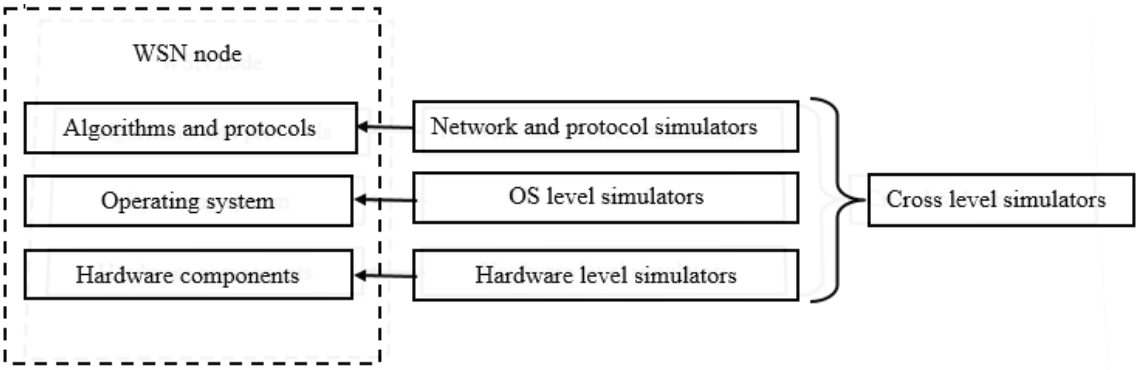


Figure 3.2 *WSN simulators targeting different WSN aspects*

3.8 Selection of simulator for WSN application development

There are tens of WSN simulators available today. Every simulator is designed with initial design goals and bundled with related features. These design goals are generally not fixed. They are kept extendible to cope the emerging needs of WSN, which simulators must follow. A simulator is usable if it provides support for new technologies and allows incorporation of new protocols and features. Simulators, which do not provide up-to-date support become unfeasible for application development and then obsolete.

The features or attributes of simulators, which an application developer can consider in selecting a feasible WSN simulator for application development are given below:

Basic algorithms, protocol and hardware support: It is necessary for simulators to provide basic set of standard protocols, algorithms, operating systems and widely used hardware platforms to facilitate WSN application development. For example, support for multiple operating systems, such as, Contiki or TinyOS will increase the usability of simulator.

Power profiling: WSN researchers, application and protocol developers are usually interested in finding how much power applications and protocols are consuming. Simulators should allow development of energy efficient protocols and algorithms by including power calculating modules.

Extendibility: New WSN protocols, procedures and hardware platforms are continuously developed and upgraded in sensor networks. Therefore, simulator should allow the development, testing and integration of new algorithms and procedures.

Performance: Performance of a simulator depends upon how simulator has been designed and how it is used. In designing simulator, high level languages, which are more efficient in performance than others are used. It is observed that C++ and Java are mostly used in developing simulator programs. While, in a simulation, number of nodes used and level of details captured affect the performance of the simulator. Higher number of nodes and greater level of details would degrade the performance and will slow down the simulator. Furthermore, poorly designed and complex protocols and algorithms bring more performance issues. A simulator should allow selection of statistics needed to be captured during simulation and it should avoid generating unnecessary data to provide an acceptable level of performance.

Scalability: Scalability of a simulator tells how many nodes can be used in a simulation with acceptable level of performance. A WSN application involves nodes that could be in the range of tens to thousands. Thus, simulators, which support required number of nodes with acceptable level of performance, are appropriate to be considered for application development.

Scripting languages: Scripting languages provide control over the simulations. They allow configuration of simulation parameters, for example, how fast or how many times a simulation should be run or how many nodes will a simulation have at particular instance of time during a simulation run. They are also used to process the output data, for example, displaying data in tables or writing output data into files.

Debugging: Debugging tools identify and resolve bugs quickly. They inspect software programs and device drivers by providing user friendly interfaces, which displays event queues, state of different variables and peripherals during execution. They also execute

instructions of program code one by one. The values of program variable can also be set at run time to observe the behavior of program during execution.

Cross level simulations support: Due to heterogeneity in WSNs, simulators should provide simulations at different levels of abstraction. They should be able to provide hardware level, software level or operating system level details in a single simulation. Because, in a network, there might be some nodes for whom their device drivers may need to be investigated while for others only performance of routing protocol or MAC protocol may be a concern.

Heterogeneous simulations support: WSN may include different sensor node applications running on different nodes in the same network. Usability of simulators increase if they have ability to simulate different node platforms and running different applications in a single simulation.

Connectivity with other simulators: WSN simulators perform specific tasks, which may or may not be present in other simulators. Thus, cooperation between simulators could be beneficial where functionality from both simulators is required in a single simulation.

Concrete documentation: The quality of technical documentation about simulator usage provide ease in using simulator for application development. A simulator with less or poorly written documentation will be difficult to use and its usage will be limited. Carefully explained example applications and core simulator constructs will help the user to quickly test and grasp the core functionalities of the simulator.

Code portability: Simulator should be able to produce compiled program code that can be easily ported to sensor node with little or no modifications.

Activity level: A simulator is considered active if it is actively developed, debugged and upgraded. Simulators, which do not provide up to date technical documentation and support become difficult to use and finally become obsolete or inactive. Simulators should have a valid, official and active website that provide details about previous and latest versions. Also, website should provide basic and advanced level details about simulator installation, usage and development. Usability of simulator can be increased by providing tutorials, user manuals, guides and example applications. Furthermore, website should provide links to developers and user community portal to facilitate bugs reporting and resolution, asking and answering queries regarding development and usage of simulators.

3.9 Simulator attributes table

A comprehensive table of simulator attributes is presented in Table 1. The table consists of the simulator attributes, which can be considered in selecting a WSN simulator for

application development. Short description for each attribute is also given. The table is constructed after analyzing existing simulators and referred articles.

Table 1. *Simulator attribute table*

No.	Simulator attribute	Short description
1	Type	General purpose or specific
2	Category	For example, network, hardware or cross level
3	Simulation Type	Discrete event or synchronous
4	Programming Language	The language in which simulator is developed
5	Version and Date	Latest version available and when it was last updated
6	License	Open source or propriety
7	Initial Design goal	Such as, extendibility
8	Widely used for	Primary use of simulator
9	Development Language	Application program programming language
10	Basic Protocol Support	Availability of standard network protocols
11	WSN Protocol Support	Availability of specific WSN protocols
12	Supported Hardware Devices	For example, Sky mote
13	Supported Operating System	For example, Contiki , TinyOS
14	Heterogeneous Support	Multiple types of nodes and applications in a simulation
15	Scalability	Number of nodes that can be simulated easily
16	Power Profiling	Support for calculation of power consumption
17	Scripting Languages	Such as, JavaScript
18	Debugger Support	Yes or no
19	Code Portability	Application code is transferable or not
20	Graphical Interfaces	Yes or no
21	Connectivity with other tools	Frameworks, which incorporates this simulator
22	Technical Documentation	Availability of detailed technical documentation
23	Technical Support	Availability of technical support
24	Example Codes	Availability of example applications by developers
25	Develop, Debug, Deploy	Development, debugging and deployment all in one
26	Research Papers	Availability of research papers, articles, surveys
27	Developers/Users Community	Availability of developers and users community
28	Activity Level of Simulator	Active or inactive
29	Extendible	Simulator is extendible or not

The WSN attributes presented in the table above can be categorized into following three groups:

- Activity attributes
- Basic attributes
- Core attributes

Activity attributes of simulator tell the activity level of simulators. Basic attributes give the information, such as, type, category, development language of simulators. Attributes

like power profiling, scalability are included in core attributes. The groups are presented in separate attribute tables Table 2, Table 3 and Table 4 below:

Table 2. *Activity attributes table*

No.	Activity attributes
1	Version and Date
2	License
3	Availability of Technical Documentation
4	Availability of Technical Support
5	Developers and users community
6	Example Codes for Application Development
7	Availability of Research Papers
8	Activity Level

Table 3. *Basic attributes table*

No.	Basic attribute
1	Type
2	Category
3	Simulation Type
4	Programming Language
5	Extendible
6	Initial Design Goal
7	Widely Used for
8	Application Development Language
9	Scripting and Other Languages
10	Graphical User Interfaces Supports
11	Debugger Support

Table 4. *Core attributes table*

No.	Core attribute
1	Basic Protocol Support
2	WSN Protocols Support
3	Supported hardware platforms
4	Supported Operating Systems
5	Heterogeneous Support
6	Scalability
7	Power Profiling
8	Connectivity with other Tools or Simulators
9	Develop, Debug, Deploy
10	Code Portability

3.10 Comparison of WSN simulators

Seven prominent WSN simulators are explored and compared using the attribute tables. These simulators were included in many comparison surveys and articles reviewed during this thesis. Table ***5***, Table 6 and Table 7 are filled up to the level of details, which are available via comparison surveys, WSN related books and articles available on the web from [55] to [65]. The documentation for Sunshine [56] and MiXim [58] are not available. It was not possible to explore them in detail. Therefore, they are not included in Table 6 and Table 7.

3.11 Feasibility of existing WSN simulators

The simulators, which were compared in comparison tables, Table 5, Table 6 and Table 7 were analyzed and their feasibilities were measured for common WSN applications requirements in Table 8. The common requirements used in analysis are given below:

1. Heterogenous network support.
2. Power profiling of node.
3. Availability of known node platforms.
4. Multi-hop routing.
5. Debugging.
6. GUI.
7. Cross level details.
8. Active simulator.
9. Basic and standard protocol support.

Since COOJA met all the requirements, it appeared to be a feasible simulator for WSN application development. COOJA measures power consumption, includes Mica [66] and Sky [67] mote platforms. COOJA is also GUI based and it allows debugging and testing of WSN applications. COOJA is also an active simulator and simulates Contiki operating system, which implements routing and communication protocols.

Table 5. *Activity attributes comparison table*

No.	Activity attribute	COOJA	TOSSIM	NS-3	OMNeTT++	Worldsens	MiXiM	Sunshine
1	Latest Version and Date	Included in: Instant Contiki 3.0, 25-Aug-15	Included in: TinyOS 2.1.2, 20 Aug 2012	NS-3.26, 3-Oct-16	OMNeT++ 5.1, 31 Mar 2017	Unavailable	MiXim 2.3 8-Mar-13	Unavailable
2	License	Open Source	Open Source	Open Source	Open Source	Open Source	Open Source	Open Source
3	Technical Documentation	Yes	Yes	Yes	Yes	Yes	No	Yes
4	Technical Support	Yes	Yes	Yes	Yes	No	No	No
5	Developers and User Community	Yes	Yes	Yes	Yes	No	Yes	No
6	Examples	Yes	Yes	Yes	Yes	Yes	No	Yes
7	Research Papers	Yes	Yes	Yes	Yes	Yes	Yes	Yes
8	Activity Level	Active	Active	Active	Active	Inactive	Inactive	Inactive

Table 6. *Basic attributes comparison table*

No.	Basic attribute	COOJA	TOSSIM	NS-3	OMNeTT++	Worldsens
1	Type	Specific	Specific	General	General	Specific
2	Category	Cross level	OS Level	Network Level	Network Level	Cross Level
3	Simulation Type	Discrete Event	Discrete Event	Discrete Event	Discrete Event	Discrete Event
4	Programming Language	Java	nesC	C++	C++	C++
5	Extensible	Yes	Yes	Yes	Yes	Yes
6	Initial Design Goal	Extendibility	Simulating TinyOS applications.	An open simulation environment for networking research	Building Network Simulations	WSN application development at multiple levels
7	Widely Used for	Heterogeneous applications development, Testing	Developing and Testing TinyOS Applications	Wireless/IP Simulations	Network Traffic, Protocol, Queuing Networks and Distributed System modeling	Information Not Available
8	Application Development Language	C	nesC	C++ and Python	Network Description Language (NED)	C++
9	Scripting and Other Lan-	JavaScript	Python, C++	Python	NED	No
10	Graphical User Interfaces	Yes	No	No	Yes	No
11	Debugger Support	Yes	Yes	Yes	Yes	Yes

Table 7. *Core attributes comparison table*

No.	Core attribute	COOJA	TOSSIM	NS-3	OMNeTT++	Worldsens
1	Basic Protocol Support	Yes	Yes	Yes	Yes	Yes
2	WSN Protocols Support	Yes	Yes	Yes	Yes (with MiXim and Castalia frameworks)	Yes
3	Supported hardware platforms	Z1, micaz, wismote, sky, ESB, native	micaz	No	No	micaz, mica2, wisenodes, sky, Telosb, Senslab nodes
4	Supported Operating Systems	Contiki, TinyOS, or Any WSN OS who produce standard ELF formatted executables	TinyOS	No	No	Any
5	Heterogeneous Support	Yes	No	Yes	Yes	Yes
6	Scalability	$<10^2$ Hardware level nodes $>10^5$ Java nodes $>3x10^4$ OS level nodes	10^3 (by default)	$>2x10^4$	$>10^4$	$>10^3$
7	Power Profiling	Yes	No	Yes	No	Yes
8	Connectivity with other	Yes	Yes	Yes	Yes	Yes
9	Develop, Debug, Deploy	Yes	Yes	No	No	Yes
10	Code portability	Yes	Yes	No	No	Yes

Table 8. *Feasibilities of simulators for WSN application development*

No.	Requirements	COOJA	NS-3	TOSSIM	WorldSens	OMNeTT++
1	Heterogenous	Yes	Yes	No	Yes	Yes
2	Power Profiling	Yes	Yes	No	Yes	No
3	Node platforms	Yes	No	Yes	Yes	No
4	Multi-hop routing	Yes	Yes	Yes	Yes	Yes
5	Debugger	Yes	Yes	Yes	Yes	Yes
6	GUI	Yes	No	No	No	Yes
7	Cross level	Yes	No	No	Yes	No
8	Active	Yes	Yes	Yes	No	Yes
9	Protocol support	Yes	Yes	Yes	Yes	Yes
	Feasibility	**Yes**	**No**	**No**	**No**	**No**

NS-3 [52] is not cross level and it does not provide GUIs. It calculates the power consumption but the power consumption calculations are generalized and not for any specific hardware platform. Hence, it was not feasible to use NS-3 for mentioned application requirements.

TOSSIM is not cross level, it is not GUI based and does not calculate the power consumption. Therefore, it is not feasible to use for application development.

OMNeTT++ [57] is not cross level and does not calculate the power consumption. Hence, it is not feasible and cannot be used for application development.

Worldsens [64][65] is a cross level simulator and it calculate power consumption, but, it is an inactive simulator. It does not provide any technical support and does not have active community of users and developers. Furthermore, it is not GUI based simulator and hence, unfeasible for application development.

4. USING COOJA FOR APPLICATION DEVELOPMENT

This chapter explores COOJA to see how it can be used to develop and test WSN applications. Thhe core constructs of COOJA, such as structure, design characteristics and Contiki OS are explained to understand the application development process on COOJA.

4.1 Structure of COOJA

COOJA supports three types of nodes in a single simulation [68]. The nodes can be from different vendors with different operating system and underlying hardware. The node types are given below:

Native nodes: Natives nodes are simulated at operating system level. Native nodes are used for developing and debugging operating system functions.

Java nodes: Java nodes are application level nodes. These nodes do not require any simulated hardware and operating system. Java nodes are used to develop and debug high level protocols and algorithms, such as, distributed algorithms.

Emulated nodes: These nodes use emulator to simulate the behavior of CPU and other hardware components, such as, radio transceivers. Contiki applications are compiled to run on emulators. Emulated nodes are used where precise hardware abstraction is required, for example, power profiling of hardware components.

Java nodes run faster than other types of nodes because they do not use any emulated hardware and operating system. However, they do not provide deployable code. Emulated nodes on the other hand are the slowest because of their dependency on operating system and emulator. [2]

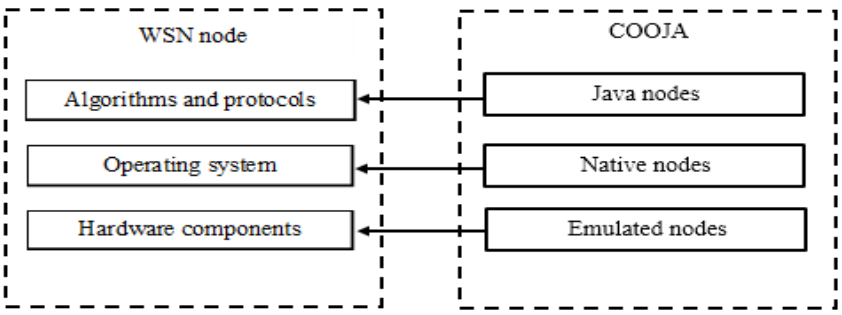


Figure 4.1 *COOJA nodes targeting node components*

4.2 COOJA design characteristics

Main components of COOJA are node type, node memory, interfaces, observers and plugins [69]. COOJA design is shown in Figure 4.2.

Node type: A simulation in COOJA may consists of different types of nodes. Nodes that run the same application program, operating system and have the same hardware components belongs to the same node type. For example, a Sky node, running hello world program over Contiki operating system is a one node type. Application program over Contiki is loaded and then compiled only once and shared among all nodes that belong to the same node type.

Node memory: Although, each node that belongs to a specific node type share the same compiled application program but the memory each node have is different and allocated separately. During the simulation, this node memory changes according to the variables of the application program. For example, a variable will be updated after a new temperature measurement is read from the simulated sensor.

Interfaces: Interfaces are used to interact with the hardware components and operating system services of node. They can be used to change the memory variables, trigger events and call the operating system functions. For example, when a button on a node is pressed via button interface, an event is generated and added in the event queue. The event is then handled and a LED can be lit via interacting with the LED interface.

COOJA observers: COOJA uses observers to listen interfaces. These functions are first registered with the corresponding interfaces and then they start listening the interfaces. There are some standard observers, for example, radio medium is a standard observer for radio transmitters of all nodes in a simulation. Radio medium listens radio transmitters, collects the data and makes it available to other transmitters expecting that data.

COOJA plugins: Plugins are graphical user interfaces, which allow user to view and edit a simulation, for example, simulation control plugin is used to start, stop and pause the simulation. COOJA main simulation window consists of number of COOJA plugins, which are shown in Figure 4.3. The short descriptions of plugins that appears in main window of COOJA are given below:

- The network window shows the simulated network of nodes. This window can be configured to show the network properties, such as, radio coverage of each node, radio traffic, IP addresses of nodes, LEDs, type and position of node. Also, the nodes can be dragged to adjust the node position and can be zoomed in and out to view the network.
- The output window displays the outputs from each node, such as ‘printf’ messages from node.

- The timeline window shows the radio events, such as radio off, radio on, radio transmission or reception in a network over time.

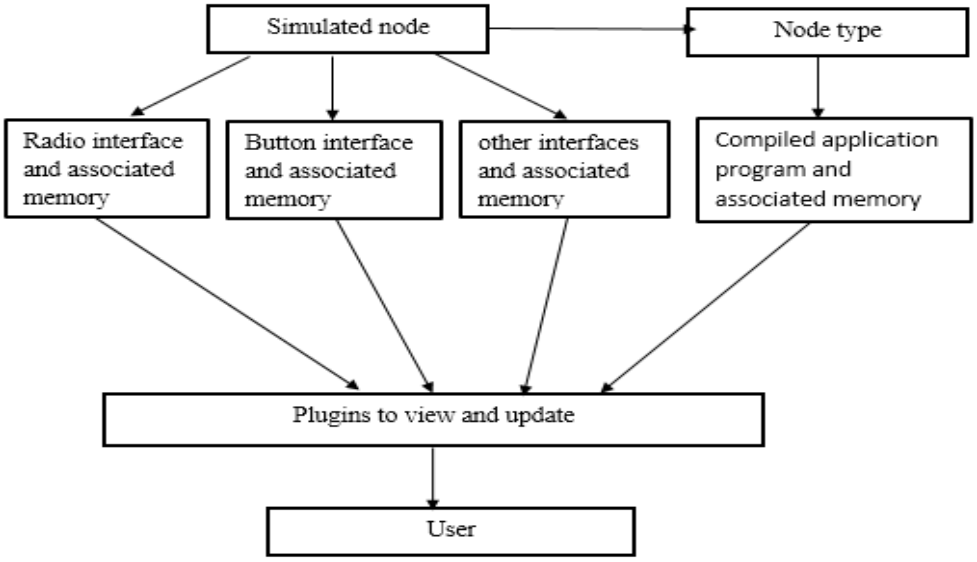


Figure 4.2 *COOJA simulator design*

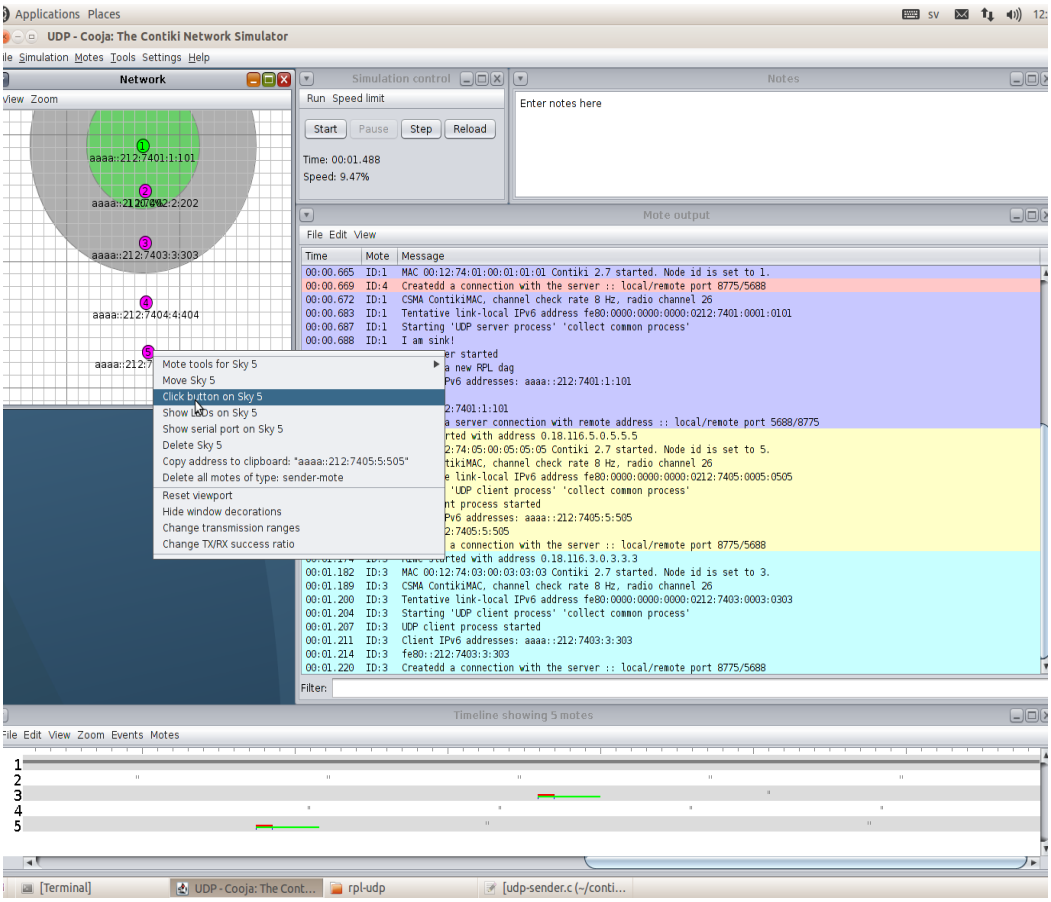


Figure 4.3 *Main simulation window*

4.3 COOJA simulation loop

COOJA interacts with all nodes one by one in a simulation loop. COOJA uses java native interface (JNI) calls to run the Contiki application programs. These calls are made from the run-time environment of COOJA to the run-time environment of Contiki to invoke the Contiki functions from COOJA. When COOJA visits or ticks a node, it first loads the memory of node and if node is awaken then it proceeds further and application program of node is run after checking the node type. Memory of the node is updated and then brought back to the java environment. COOJA then moves to the next node. At the end, the simulation time is updated and loop starts again as shown in Figure 4.4.

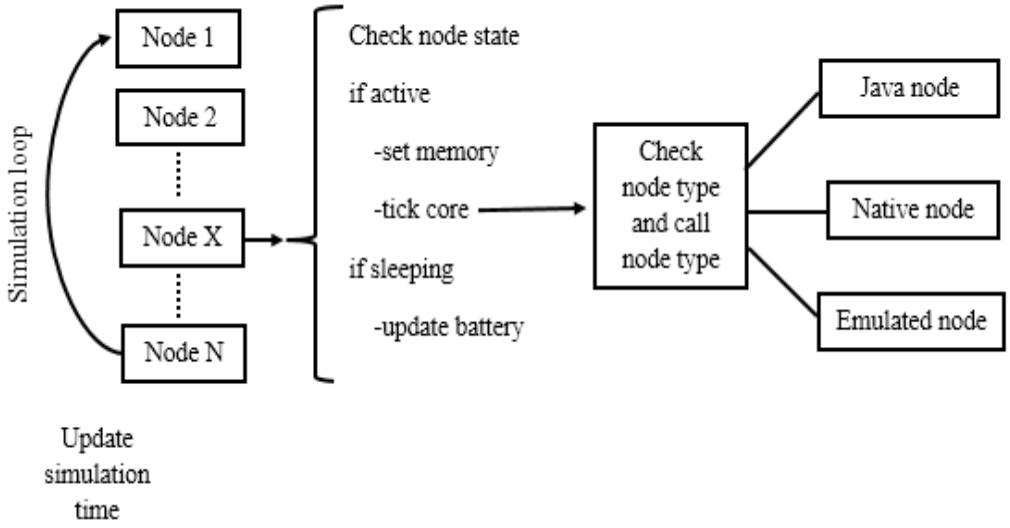


Figure 4.4 *COOJA simulation loop*

4.4 Steps in creating a new COOJA simulation

When simulation is started, all nodes are powered up one by one and Contiki is started. Nodes are configured and initialized with the default settings for the mote type. This includes, for example, assignment of mote ID, type of MAC protocol to be used, assignment of IP addresses and radio channel frequency. The default Contiki configuration can be changed by updating the project configuration file. The steps in creating a simulation in COOJA are given below:

- A new simulation in COOJA is created by clicking *New Simulation* in file menu. A new window appears and asks for the title of simulation and basic simulation

parameters, such as radio medium to be used and random start up delay time of nodes. Simulation is then created by pressing create button.

- The main simulation window appears, which shows different plugins to interact with the simulation, such as network window and simulation control window. The node type is created from the menu "Motes". Note that COOJA uses the term "mote" to refer to a node. This menu displays the number of the motes type available that can be used in a simulation, for example, emulated mote, operating system level mote and java mote. Number of specific hardware platforms are also included, such as, Sky mote, Mica mote. After selection of a mote type, Contiki application is selected for this mote type. The process is compiled for this specific mote type using built-in compiler of COOJA.
- After creating a mote type, number of motes along with their position coordinates are selected and added in the simulation.
- The process of creating node type is repeated until all the required node types are created and added in the simulation.
- Simulation is run by pressing start button on simulation control window.

COOJA simulation development cycle is shown in Figure 4.5.

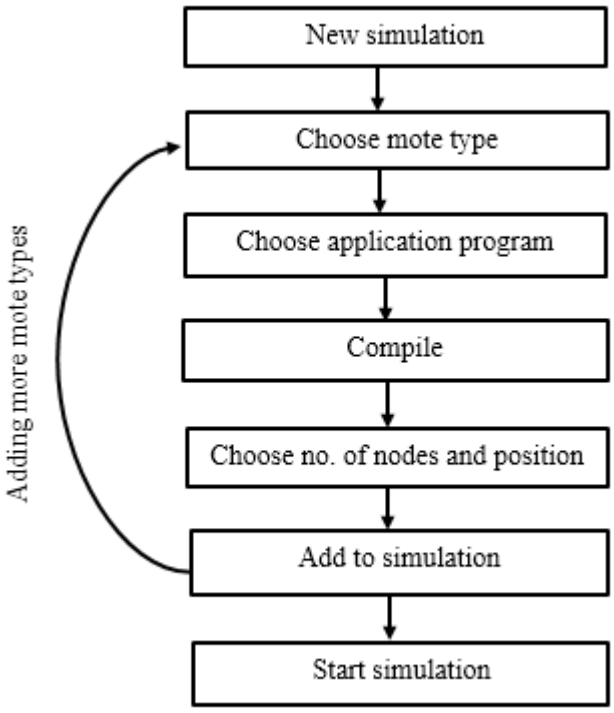


Figure 4.5 *Simulation development cycle*

The simulation can also be saved and reopened. COOJA saves all mote types, number and placement of nodes in a simulation. The running state is not saved and simulation starts from the beginning whenever simulation is reopened and started.

4.5 COOJA tools

COOJA presents set of tools to observe, collect and edit the simulation outcomes or statistics. These tools can be accessed from the menu “Tools” of the main simulation window. Short description about these tools are given below:

Collect-view: This tool is used to collect different kind of metrics, such as, power metrics, network metrics, sensor values.

PowerTracker: The tool is used to observe radio duty cycle of each node present in simulation. It tells how much each transmitter or receiver is active during an on-and-off cycle during simulation.

Radio messages: This tool is used to view the radio messages sent and received with respect to time. It also shows the content of the messages.

Buffer view: Contiki has different kind of memory buffers for its internal processing, such as, packet buffer, where packets are stored before transmission and after reception. COOJA allow inspection of such buffers with a tool called buffer view.

Breakpoints: Breakpoints are used to temporarily stop the simulation for inspection. A simulation is paused when some events, such as, radio events happen or state of some mote interface changes.

Mote interfaces viewer: This tool is used to view mote interfaces, such as button interface, LED interface.

Mote variable watcher: This tool is used to watch Contiki and application program variables and their values.

MSPSim emulator: The tool provides command line interface to interact with the mote hardware components. Also, the instructions that run on the emulator can be observed. The tool also provides graphical representation of memory stack of mote.

These tools are registered with the simulation and then they start collecting simulation statistics. Thus, they only collect statistics when needed and do not collect the unnecessary and unwanted statistics and make COOJA simulation to run faster.

4.6 Contiki overview

COOJA simulates Contiki applications [70]. It was noticed during application development that in order to develop Contiki applications, the knowledge of Contiki is required. The main features of Contiki are listed below:

- Open source operating system for WSN nodes.

- Event based kernel.
- Loads modules dynamically.
- Consumes memory resources efficiently.
- Includes standard network protocols, such as, TCP, UDP, IPv4, IPv6, RPL.
- Includes power saving and simple MAC protocols, such as, ContikiMAC
- Provides power estimations.
- Supports stack less lightweight threads, called, protothreads.
- Includes file system for reading, writing on external flash.
- Includes a networking stack for simple network operations, Rime stack.
- Can be run on wide range of platforms, such as, Sky and Mica.
- Provides command line shell for application development and debugging.
- Includes example applications to understand application development using Contiki.

4.7 Contiki application

A Contiki application is a program that runs on node hardware. A Contiki application consists of "processes", which are written in C language. A complete Contiki application is created by following the steps given below:

Process declaration: Each process is declared by using a macro `PROCESS`. It takes two parameters. The first parameter specifies the name while the second gives name of a process in string format:

```
PROCESS(hello_world_process, "A hello world process");
```

Processes to include: This macro specifies processes, which are to be executed in a Contiki application. The macro takes reference to processes as parameters, for example:

```
AUTOSTART_PROCESSES(&process_one, &process_two);
```

Process definition: This macro is used to define a process. It takes the name of a process as a parameter. It also takes two more parameters, *ev* and *data*. The parameter *ev* refers to the event that will be posted to the process while *data* specifies the data, which comes with the posted event:

```
PROCESS_THREAD(hello_world_process, ev, data) ;
```

Process begin and end: This section marks the start and end of process thread code. All code that belongs to this process thread resides in between process begin and end:

```
PROCESS_BEGIN()
printf("Hello, world!\n");
PROCESS_END();
```

Event handling: Events are handled by event handling macros in a continuously running while loop. These macros block the running process and wait for the event to happen. When an event arrives, the macros allow the process execution and event handling code is executed:

```
while (1) {
PROCESS_WAIT_EVENT ();
if(ev == sensors_event && data == &button_sensor) {
leds_toggle(LEDS_ALL);
}
```

The code presented above waits for the button press event and toggle all LEDs when the button is pressed.

Static local variables: Local variables defined in a process thread are stored in run-time process stack. Their values are not available in between process block calls, such as the PORCESS_WAIT_EVENT call, which waits for an event and blocks the process execution. The content of stack is not saved and stack is reused during each call. Hence, local variables are declared static to ensure that local variables values remain intact. Static variables are stored in data segment of the process instead of process stack and remain available during block calls.

Timers: Contiki supports different types of timers for application development, such as event timer, which generates an event when it is expired or real-time timer, which is used to call functions at exact time. An event time (etimer) is declared, initialized and used in a following way:

```
static struct etimer et;  /*timer declaration*/
etimer_set(&et, CLOCK_SECOND*6); /*timer initialized and last for 6 seconds */
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
```

Energest module: The energest module is used to calculate the power consumption. Energest module is part of Contiki code and measures the time during which each component remains ON. The measured time is in the form of timer ticks. The calculated time is then used to measure per component power consumption according to the data sheet provided by the manufacturer of hardware component. For example, energest is used to calculate time consumed by radio transmitter in a following way:

```
static unsigned long tx_start_time, tx_end_time, tx_on_time;
tx_start_time  = energest_type_time(ENERGEST_TYPE_TRANSMIT);
// put code here
tx_end_time  = energest_type_time(ENERGEST_TYPE_TRANSMT);
tx_on_time = tx_end_time  - tx_start_time;
```

The power can be calculated by using following formula:

$$P\ (mW) = \frac{consumed\ timer\ ticks \times I \times V}{timer\ ticks\ per\ second} \quad (1)$$

Where P denotes the power consumed by a hardware component in milliwatts, 'I' means current and 'V' is voltage drop across hardware component.

A complete Contiki application is given below in Program 1:

```
#include "contiki.h"
#include "dev/button-sensor.h"
#include "dev/light-sensor.h"
#include "dev/leds.h"
#include <stdio.h> /* for printf*/
/*Process Creation*/
/*---------------------------------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------*/
/*Process Definition*/
/*---------------------------------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data)
{
   /*Process thread*/
   PROCESS_BEGIN();
   /*Local varibles and sensor activation */
   static uint8_t counter = 0;
   SENSORS_ACTIVATE(button_sensor);
   SENSORS_ACTIVATE(light_sensor);
   /*reading and priting light sensor values*/
   printf("Hello, World \n");
   printf("Light: \%u\n", light_sensor.value(0));
   /*Never ending while loop*/
   while(1)
      {
        /*wait for the button sensor event */
            PROCESS_WAIT_EVENT();
            if(ev == sensors_event && data == &button_sensor)
            {
            /*increment and print the counter,  */
                  counter++;
                  printf("counter: \%u\n", counter);
                  leds_invert(LEDS_ALL);
            }
    }
  PROCESS_END();
}
/*---------------------------------------------------------------------*/
```

Program 1. *Contiki application*

4.8 COOJA and Contiki installation

For quick application development, the developers of COOJA and Contiki, provide a development environment that includes Contiki, COOJA and all the relevant development tools, such as, compilers, emulators and debugger. The developer environment, which comes as an Ubuntu Linux virtual machine is called “Instant Contiki” and runs in virtual machine player, such as VMWare. Instant Contiki and VMWare can be download from Contiki and VMWare [71] official websites. After running Ubuntu virtual machine from VMWare, COOJA is run by using the following commands from Ubuntu terminal:

```
cd contiki/tools/cooja
ant run
```

COOJA files are compiled and then initial COOJA main screen appears.

5. TEST RESULTS AND FEASABILITY ANALYSIS OF COOJA

The feasibility of COOJA was measured experimentally by designing three test applications. These ensure that necessary functionality can be achieved and statistics collected from COOJA for the WSN application development. The application requirements were based on typical WSN application functionality presented in Chapter 2. The set requirements were a typical WSN application with multi-hop routing, calculation of CPU execution time and power consumption of the application program code.

5.1 COOJA test application I

The first Contiki application was designed to test how COOJA collects the simulation statistics for an IPv6 and UDP based multi-hop network and what kind of statistics does COOJA capture during simulation.

Description: This application used Sky node and comprised of two types of nodes, sensor nodes and one sink node. Sensor nodes measured the light sensor value and sent it to the sink node via intermediate nodes after button click. Sink node sent an acknowledgment to the sending node that the data has been received successfully. The application used RPL multi-hop routing protocol, UDP protocol for data transmission and IPv6 protocol for network addressing.

Requirements: The following requirements were set for application 1:

1. RPL for multi-hop routing
2. IPv6 and UDP based network
3. Reading light sensor values
4. Default Contiki MAC

COOJA simulation: The simulation designed for this application involved five sky nodes and followed a simple tree topology. Sink node used ID 1 for identification, while the other four sensor motes used IDs 2, 3, 4, 5 respectively. Both sink and sensor nodes executed different Contiki processes.

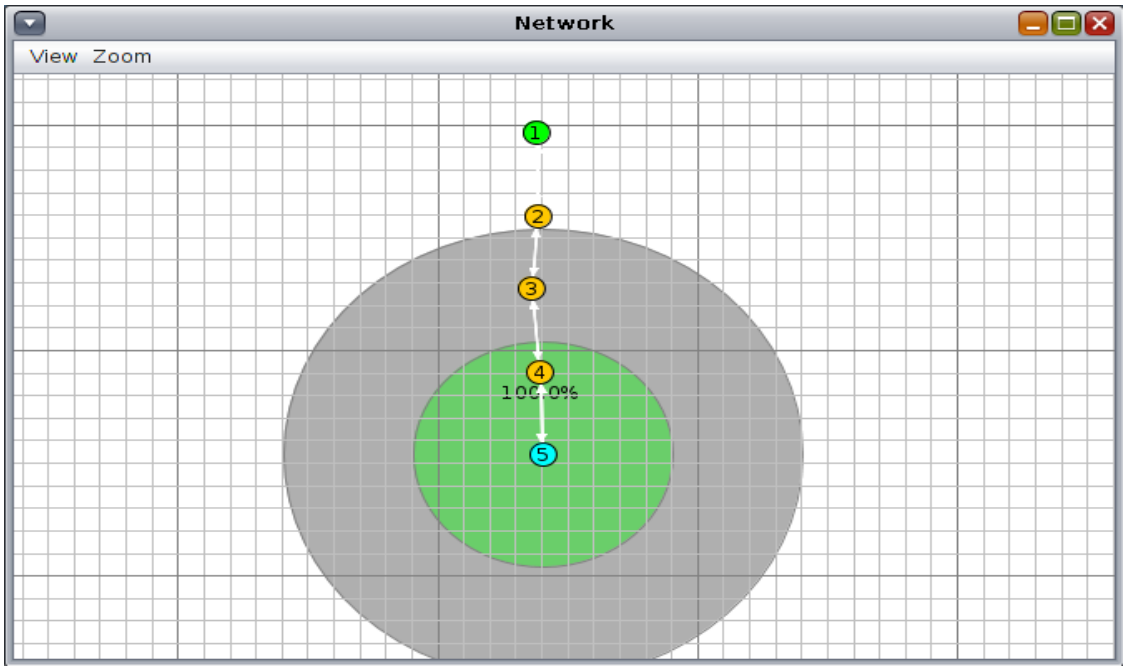


Figure 5.1 *Test application I, topology and coverage*

Simulation results: The simulation results were captured in the network, mote output and collect-view windows.

Result 1: Network window displayed the transmission of UDP messages between node 5 and node 1. This window also allowed interaction with the node. For example, in this simulation, a button click event was generated from node 5 by using network window. This window also showed topology and area coverage of node as shown in Figure 5.1.

Result 2: Node processes were programmed to report UDP communication and light sensor value events on button click as shown in Figure 5.2. On button click, sensor node 5 measured the light sensor value and sent it to the sink node 1 in a UDP message. Sink received the message via intermediate nodes and sent the acknowledgement back to node 5. By default, COOJA senses the light as random integer values.

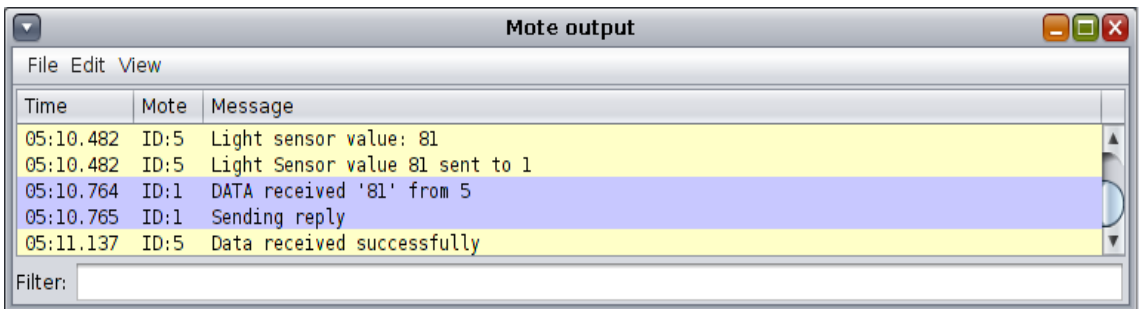
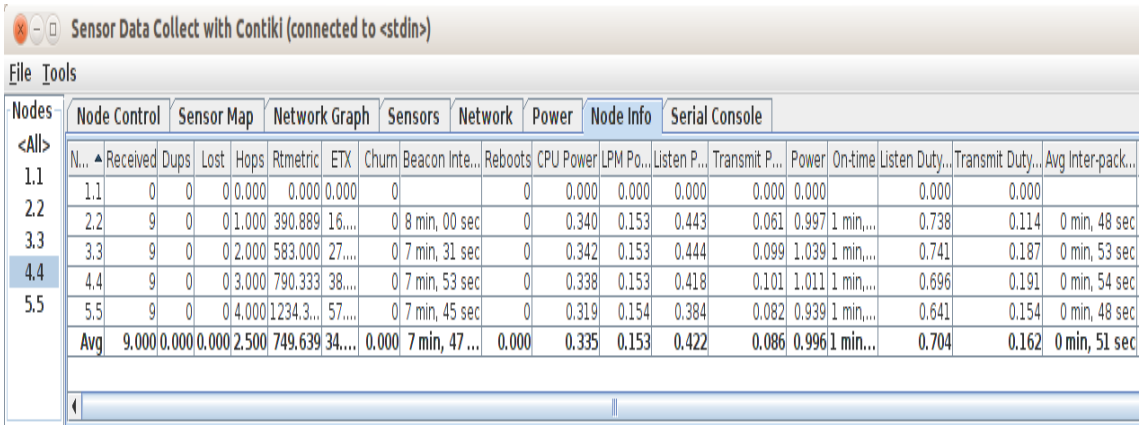


Figure 5.2 *Output window*

Result 3: Simulation statistics were captured in collect-view window. It was observed that COOJA captured four type of simulation statistics during simulation as shown in Figure 5.3 and Figure 5.4.

The captured statistics are given below:

- **Power statistics:** such as average radio transmit power consumption per node.
- **Network statistics**: such as network hops per node, packet received per node.
- **Sensor statistics:** such as light sensor values, battery voltage.
- **Topology statistics:** such as sensor map, which shows the best routing path available between the nodes along with the path quality values.



N...	Received	Dups	Lost	Hops	Rtmetric	ETX	Churn	Beacon Inte...	Reboots	CPU Power	LPM Po...	Listen P...	Transmit P...	Power	On-time	Listen Duty...	Transmit Duty...	Avg Inter-pack...
1.1	0	0	0	0.000	0.000	0.000	0		0	0.000	0.000	0.000	0.000	0.000		0.000	0.000	
2.2	9	0	0	1.000	390.889	16....	0	8 min, 00 sec	0	0.340	0.153	0.443	0.061	0.997	1 min,...	0.738	0.114	0 min, 48 sec
3.3	9	0	0	2.000	583.000	27....	0	7 min, 31 sec	0	0.342	0.153	0.444	0.099	1.039	1 min,...	0.741	0.187	0 min, 53 sec
4.4	9	0	0	3.000	790.333	38....	0	7 min, 53 sec	0	0.338	0.153	0.418	0.101	1.011	1 min,...	0.696	0.191	0 min, 54 sec
5.5	9	0	0	4.000	1234.3...	57....	0	7 min, 45 sec	0	0.319	0.154	0.384	0.082	0.939	1 min,...	0.641	0.154	0 min, 48 sec
Avg	9.000	0.000	0.000	2.500	749.639	34....	0.000	7 min, 47 ...	0.000	0.335	0.153	0.422	0.086	0.996	1 min...	0.704	0.162	0 min, 51 sec

Figure 5.3 *Collect-view window*

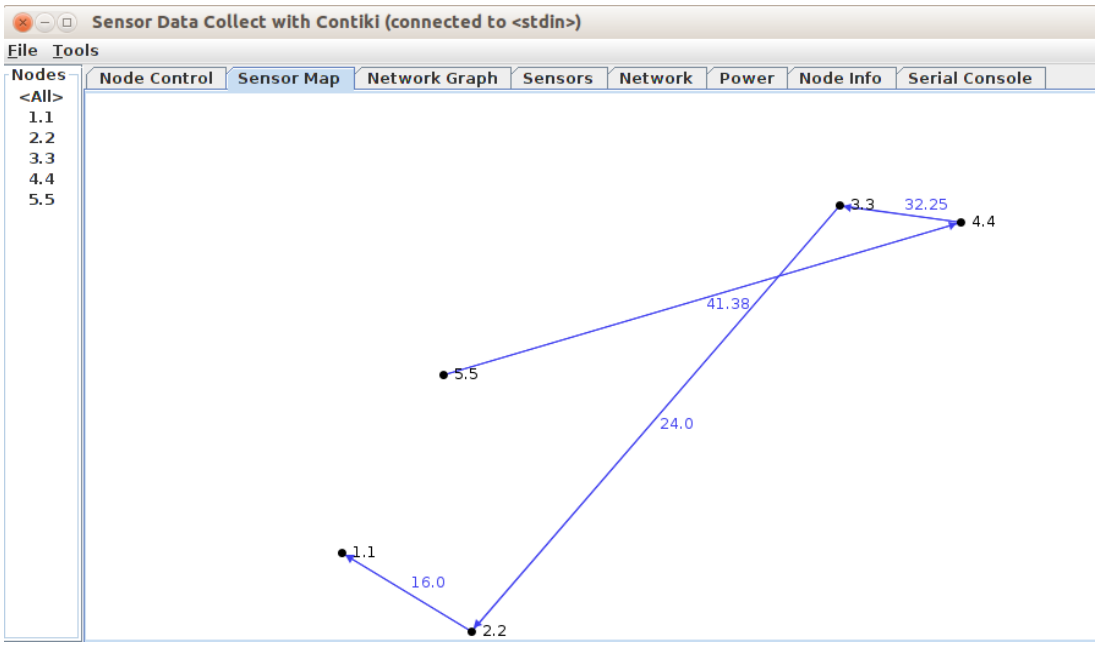


Figure 5.4 *Sensor map window*

COOJA feasibility analysis:

COOJA was able to simulate an IPv6 and UDP based network with RPL multi-hop routing. These protocols were provided by Contiki operating system, which COOJA simulates. COOJA also provided four type of statistics to study network, power, topology and sensor characteristics in a simulation. COOJA interfaces allowed interaction with the simulation and nodes visually. The communication and content of the messages were also observable in COOJA windows.

COOJA presented appropriate number of statistics for this application and provided visual control over simulation. Therefore, COOJA is feasible to develop and test WSN applications, which require IPv6 and UDP based multi-hop networks.

Other remarks for the test: Contiki services were required in application development on COOJA. Also, some of the COOJA tools could not be used if they were not handled in and called from the application process, such as collect-view. In order to use collect-view, a message is constructed in an application process and consists of network and power statistics. This seems to be an efficient approach that COOJA tools only collect necessary statistics when the tools are told by the application process. But, on the hand, it also requires sufficient knowledge of how to call and handle COOJA tools from an application process. Note that collect-view window shows IDs of nodes in different format. For example, node with ID 1 will be shown as 1.1. Furthermore, the sensor map window shows the nodes in different layout than the layout, which is shown in network window. The reasons for such variations are not documented and are unknown. Also, collect-view module is not properly commented and is difficult to understand and use in the beginning.

5.2 COOJA test application II

The second Contiki application was designed to test how COOJA can be used to calculate the power and memory consumption of a node.

Description: Artificial neural networks (ANN) are complex computational models and used in computer sciences and research disciplines. They have high processing and memory demands because of an iterative self-learning process. ANN learns by matching the classification of input data with the classification of known data. The errors found into initial classification helps in adjusting the ANN algorithm and process is repeated.

ANN code model was ported for Contiki and COOJA. The model represented basic tasks of ANN, but in a simplified and restricted in manner without using back-propagation and activation function with simple threshold summation of weights. A comparison was made

between Contiki application with ANN code execution and another Contiki application without ANN code execution.

Requirements: The following calculations were required from this application:

1. Calculation of CPU execution time.
2. Calculation of CPU power consumption.
3. Calculation of memory consumption.

COOJA simulation: Two Sky nodes were used. The sending node executed a print statement "Helloo world", calculated its execution time and sent the statistics to sink node. While, on button click, the sending node executed the ANN model code, calculated execution time and statistics were sent to sink node again.

Simulation results: Simulation results were captured in the mote output, collect-view, and MSP stack watcher windows.

Result 1: CPU execution time for printing "Helloo world" only took 28 CPU timer ticks while ANN code consumed 1054 ticks as shown in Figure 5.5 and Figure 5.6.

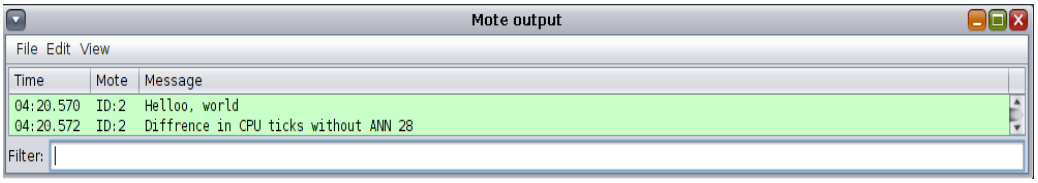


Figure 5.5 *Mote output without ANN*

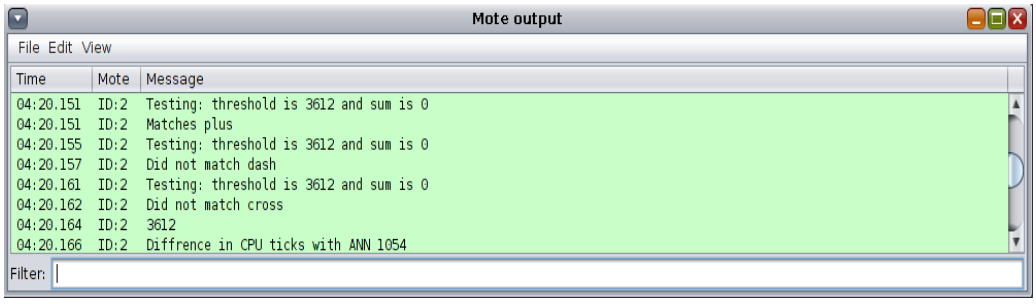


Figure 5.6 *Mote output with ANN*

Result 2: CPU power consumption was up to 0,35mW without ANN execution but drastically increased up to 5,30mW after ANN execution as shown in Figure 5.7 and Figure 5.8.

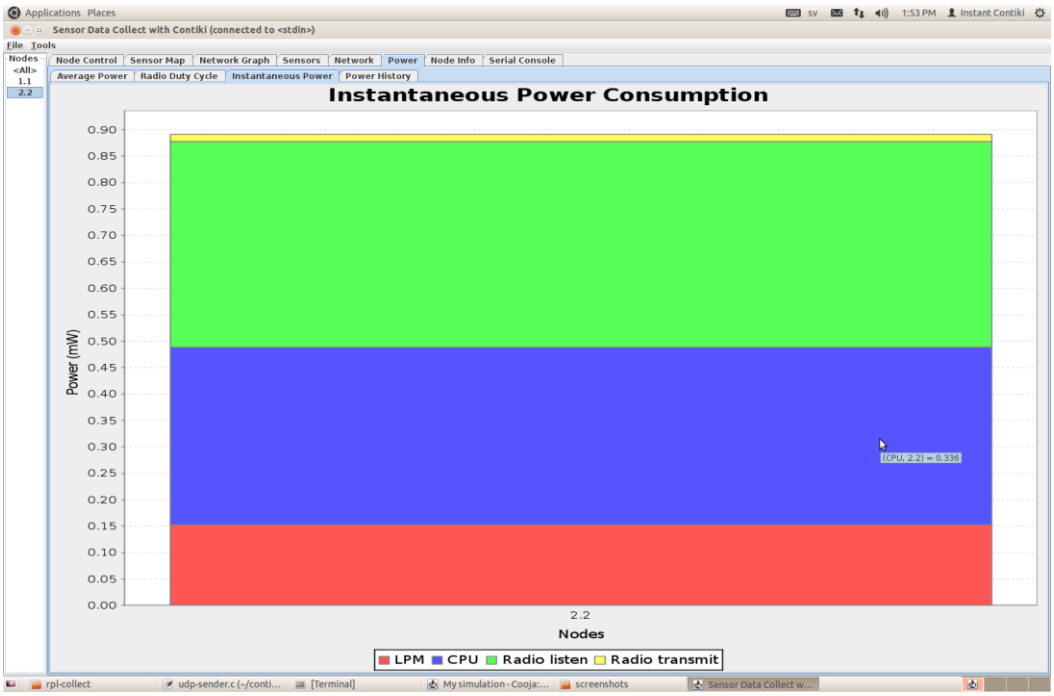


Figure 5.7 *Instantaneous power consumption without ANN execution*

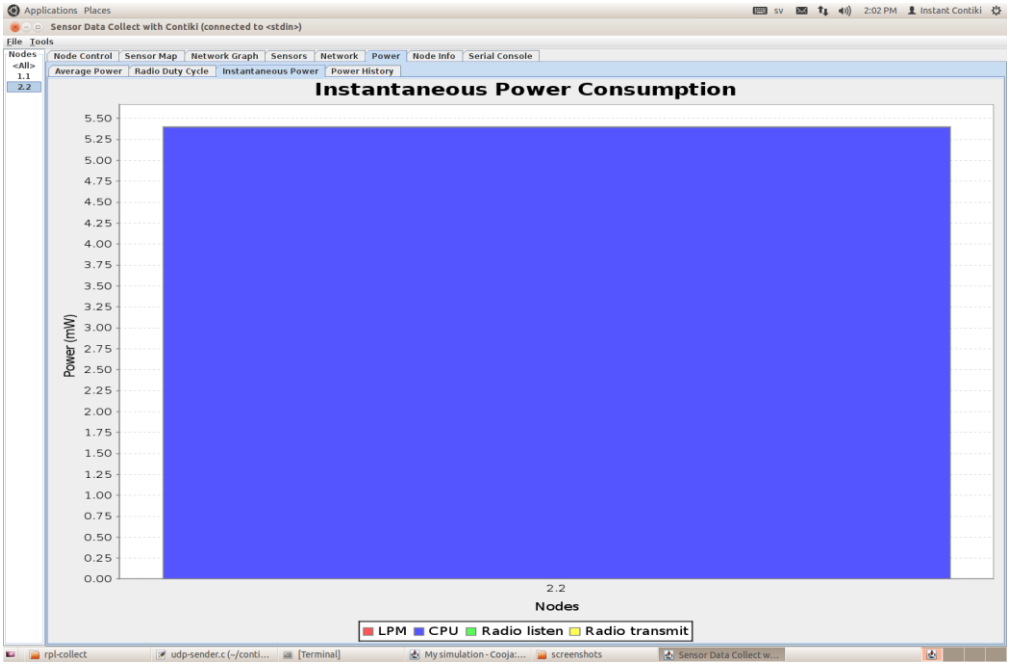


Figure 5.8 *Instantaneous power consumption with ANN execution*

Result 3: Although, stack watcher did not mention the unit of measurement in the graph but high spikes in the graph were observed during ANN execution Figure 5.9.

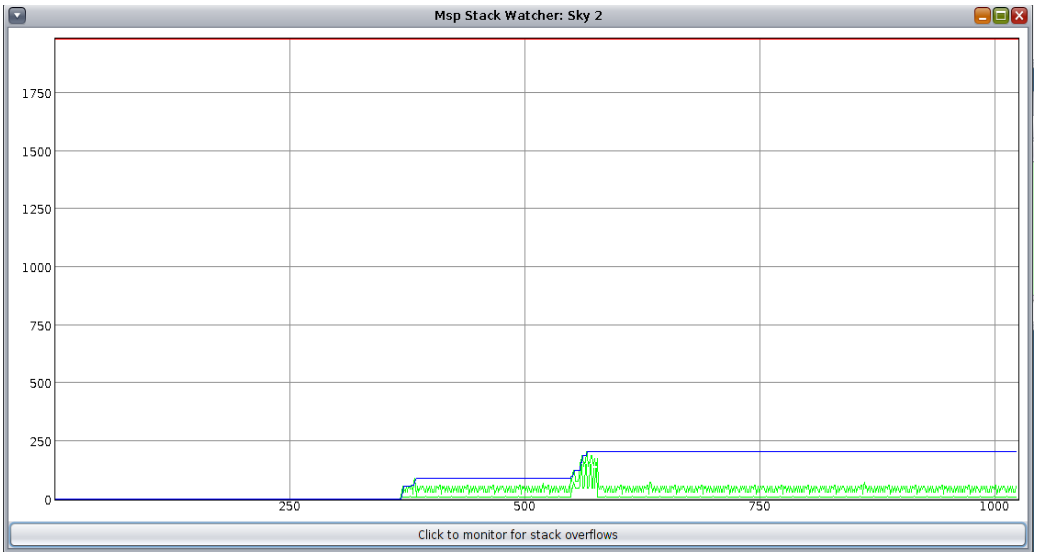


Figure 5.9 *Memory stack with ANN*

COOJA feasibility analysis: COOJA was able to measure CPU execution time, CPU power consumption and memory consumption of node. However, the presented graph of memory stack did not display the units of memory consumption and concluding memory results was difficult. However, COOJA fulfilled its requirements and is feasible to be used for development of WSN applications, which require CPU execution time, CPU power consumption and memory measurements.

Other remarks for the test: It was observed that collect-view could not be used with all node platforms included in COOJA. For example, collect-view tool did not work with Mica platforms.

5.3 COOJA test application III

The third application was designed to test COOJA capability of calculating the CPU power consumed by different code blocks or functions**.**

Description: ANN model presented in previous application was reused. The CPU power consumption was calculated according to Sky mote data sheet [67] before and after each function call of ANN model code. Operational voltage of Sky mote is around 3 Volts and current through CPU during active time is 1.8 mA. Real-time ticks are 32,768 per second.

Requirements: Following requirements were set for this application:

1. CPU execution time calculation.
2. CPU consumed power calculation.
3. Use of Sky mote datasheet.

Simulation scenario: One Sky mote was used during this simulation. ANN code was executed and CPU power was calculated after button click.

Simulation results: The simulation results were collected in mote output window shown in Figure 5.10.

```
00:15.344  ID:1   ================================================================================================
00:15.362  ID:1   difference in timer ticks in executing teaching function 449
00:15.366  ID:1   Teaching funtion Execution time  0.013702 seconds
00:15.371  ID:1   CPU ticks used for ANN_ann_teach_correct_plus function execution 449
00:15.373  ID:1   power in mW 0.07399
00:15.377  ID:1   lpm ticks used for ANN_ann_teach_correct_plus 0
00:15.384  ID:1   ================================================================================================
00:15.390  ID:1   difference in timer ticks in executing test funtion for plus input 25
00:15.394  ID:1   test funtion for plus input Execution time  0.000762 seconds
00:15.396  ID:1   Matches plus
00:15.400  ID:1   CPU ticks in executing test funtion for plus input 23
00:15.402  ID:1   power in mW 0.00379
00:15.406  ID:1   LPM ticks used for test funtion for plus input 0
00:15.412  ID:1   ================================================================================================
00:15.420  ID:1   difference in timer ticks in executing test funtion for dash input 23
00:15.425  ID:1   Test function for dash input Execution time  0.000701 seconds
00:15.427  ID:1   Did not match dash
00:15.431  ID:1   CPU ticks in executing test funtion for dash input 23
00:15.432  ID:1   power in mW 0.00379
00:15.436  ID:1   lpm ticks used for test funtion for dash input 0
00:15.443  ID:1   ================================================================================================
00:15.449  ID:1   difference in timer ticks in executing test funtion for cross input 22
00:15.454  ID:1   Test function for cross input Execution time  0.000671 seconds
00:15.456  ID:1   Did not match cross
00:15.460  ID:1   CPU ticks in executing test funtion for cross input 23
00:15.462  ID:1   power in mW 0.00379
00:15.466  ID:1   LPM used for test funtion for cross input  0
00:15.472  ID:1   ================================================================================================
```

Figure 5.10 *ANN functions execution ticks and power consumption*

Result 1: ANN teaching function consumed 449 timer ticks during its execution and calculated consumed power was 0,07399mW. Whereas, test functions consumed comparatively less timer ticks than teaching function and were in the range of 22 to 25 timer ticks per test function.

COOJA feasibility analysis: COOJA was able to measure execution time and power consumption of each code block precisely. Application used energest module of Contiki operating system and MSPsim emulator to calculate the execution time of different code blocks precisely. CPU power was calculated by putting the execution time and Sky node datasheet values in equation 1. Hence, COOJA is feasible to be used for development of WSN applications, which require calculations of CPU execution time and consumed power on Sky mote precisely.

Other remarks for the test: MSPsim emulates Sky mote. MSPSim also provides a command line interface to interact with each running node. However, the user guide for MSPsim is not available and hence makes it difficult to use by the command line interface.

Concluding Remarks for COOJA

The feasibility of COOJA was determined by using three different test applications. The test applications were designed with the most common requirements found in typical WSN applications. The requirements were power and memory consumption calculations and multi-hop routing on IPv6 and UDP based networks. COOJA fulfilled all the requirements and is feasible to use for WSN application development. However, COOJA have limitations as well, which are given below:

1. COOJA requires knowledge of Contiki operating system for application development on COOJA.
2. All COOJA tools do not work with all node platforms, which are available in COOJA.
3. Collect-view is not well documented and is difficult to understand and use.
4. The documentation for using MSPsim emulator from command line interface is not available and hence, makes it difficult to use.

6. CONCLUSION

WSNs have applications in various fields of science and technology that are diverse in nature. WSN simulators are used to develop and test WSN applications to speed up and reduce costs. WSN applications are diverse and simulators are used to develop and test WSN applications. Selection of a feasible WSN simulator is foremost requirement before application development may begin. Feasible simulators fulfil application requirements and allow low-cost development and testing of WSN applications in short time.

This thesis analyzed WSN simulators and presented the simulator attributes, which a WSN application developer should consider in selecting a feasible simulator for WSN application development. Three types of attributes were collected. The first type specified activity level of simulator. In the second type, basic attributes such as type, category, development language of simulators were collected. In the third type, core WSN attributes like power profiling and scalability were gathered. These attributes were used in comparing seven prominent simulators and COOJA was selected as a feasible simulator for application development. COOJA had features that matched the common requirements of typical WSN applications. COOJA was also explored experimentally to test its feasibility for application development. The requirements for testing feasibility were development of typical WSN applications with multi-hop networking, calculation of CPU execution time and power consumption.

Three test applications were designed with the requirements in COOJA. In the test application I, COOJA was able to simulate an IPv6 and UDP based multi-hop network. COOJA captured four types of statistics during simulation of this application. The statistics were network, power, sensor and topology statistics. In the test application II, COOJA calculated CPU power and memory consumption of ANN model code. Collect-view tool was used to collect power statistics and MSPsim stack watcher was used to observe memory consumption. In the third application, CPU execution time and CPU power consumption for each code block of ANN model was measured. COOJA was able to calculate precise execution time by using MSPsim emulator. COOJA also calculated CPU power consumption by using Sky mote datasheet.

COOJA successfully met all the requirements and provided visual control over simulation. Hence, COOJA was found out as a feasible simulator for WSN application development. However, application development on COOJA can be enhanced by adding support for more hardware platforms. Also, the usability of COOJA can be improved by adding documentation about collect-view and MSPsim emulator.

REFERENCES

[1] Patrik Moravek, Dan Komosny, Milan Simek Brno - Specifics of WSN simulations, Elektrorevue Vol.3, No. 3, 2011, pp. 1-7, Available: www.elektrorevue.cz/en/download/specifics-of-wsn-simulations

[2] Joaksim Eriksson - Detailed Simulation of Heterogeneous Wireless Sensor Networks, Licentiate Thesis, Department of Information Technology, Uppsala University, 2009, Available: https://uu.diva-portal.org/smash/get/diva2:388081/FULLTEXT01.pdf

[3] M. M. N. Aldeer, A summary survey on recent applications of wireless sensor networks, IEEE Student Conference on Research and Development, Putrajaya, 2013, pp. 485-490, Available: doi: 10.1109/SCOReD.2013.7002637

[4] K. Langendoen, A. Baggio and O. Visser, Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture, Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, 2006, pp. 8, Available: doi:10.1109/IPDPS.2006.1639412

[5] Miloš Jevtić, Nikola Zogović, Evaluation of Wireless Sensor Network Simulators, 17th Telecommunications forum TELFOR 2009 Serbia, Belgrade, pp. 1-4, 2009, Available: http://2009.telfor.rs/files/radovi/10_48.pdf

[6] TOSSIM, Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM, Accessed: 2017-02-25

[7] NS-2, Available: http://nsnam.sourceforge.net/wiki/index.php/User_Information, Accessed: 2017-02-25

[8] COOJA and Contiki, Available: www.contiki-os.org/start.html, Accessed: 2017-02-25

[9] Mrs. Poonam Chhimwal, Dhajvir Singh Rai , Deepesh Rawat, Comparison between Different Wireless Sensor Simulation Tools, Journal of Electronics and Communication Engineering (IOSSR - JECE), 2013, Vol. 5, Issue 2, pp. 54-60, Available: http://article.sapub.org/10.5923.j.jwnc.20150501.03.html

[10] Anand Nayyar, Rajeshwar Singh, A Comprehensive Review of Simulation Tools for Wireless Sensor Networks (WSNs), Journal of Wireless Networking and Communications, 2015, pp. 19-47, Available: http://article.sapub.org/10.5923.j.jwnc.20150501.03.html

[11] E. Egea-López, J. Vales-Alonso, A. S. Martínez-Sala, P. Pavón-Mariño, J. García-Haro, Simulation Tools for Wireless Sensor Networks, Summer Simulation Multi-conference, 2005, Available: http://ait.upct.es/~eegea/pub/spects05.pdf

[12] Mauri Kuorilehto, Mikko Kohvakka, Jukka Suhonen, Panu Hämäläinen, Marko Hännikäinen, Timo D. Hamalainen, Ultra-Low Energy Wireless Sensor Networks in Practice: Theory, Realization and Deployment, Wiley, 2007, pp. 51-63

[13] Cecilio, José, Furtado, Pedro, Wireless Sensors in Heterogeneous Networked Systems Configuration and Operation Middleware, Springer, 2014, pp. 5-25

[14] António Grilo, Wireless Sensor Networks Chapter 2: Single node architecture, Information and Communication Technology Institute, 2007, pp. 4, Available: http://comp.ist.utl.pt/ece-wsn/doc/slides/sensys-ch2-single-node.pdf, Accessed: 2017-03-05

[15] Thang Vu Chien, Hung Nguyen Chan, and Thanh Nguyen Huu, A Comparative Study on Operating System for Wireless Sensor Networks, International Conference on Advanced Computer Science and Information Systems, 2011, pp. 73-78, Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6140770&isnumber=6140728

[16] Y. Zhang and X. w. Guo, Design and Implementation of Wireless Sensor Network Node Based on .Net Micro Framework, International Conference on Internet Technology and Applications, Wuhan, 2010, pp. 1-3, Available: doi: 10.1109/ITAPP.2010.556629

[17] Prabal Dutta, Adam Dunkels, Operating Systems and Network Protocols for Wireless Sensor Networks, 2011, Available: DOI: 10.1098/rsta.2011.0330

[18] W. Schott, A. Gluhak, M. Presser, U. Hunkeler and R. Tafazolli, e-SENSE Protocol Stack Architecture for Wireless Sensor Networks, 2007, 16th IST Mobile and Wireless Communications Summit, pp. 1-5, Available: doi: 10.1109/ISTMWC.2007.4299121

[19] S. Hadim and N. Mohamed, Middleware: middleware challenges and approaches for wireless sensor networks, IEEE Distributed Systems Online, vol. 7, no. 3, 2006, pp. 1-1, Available: doi: 10.1109/MDSO.2006.19

[20] Jolly Soparia, Nirav Bhatt, A Survey on Comparative Study of Wireless Sensor Network Topologies, Internal Journal of Computer Applications, 2014, pp. 40-43, Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.428.5507&rep=rep1&type=pdf

[21] Divya Sharma, Sandeep Verma, Kanika Sharma, Network Topologies in Wireless Sensor Networks: A Review, IJECT, 2013, pp. 93-97, Available: http://www.iject.org/vol4/spl3/c0116.pdf

[22] Gurwinder Kaur and Rachit Mohan Garg, Energy Efficient Topologies for Wireless Networks, International Journal of Distributed and Parallel Systems (IJDPS), 2012, pp. 179-192, Available: http://airccse.org/journal/ijdps/papers/3512ijdps16.pdf

[23] Pilloni Virginia , Dynamic Deployment of Applications in Wireless Sensor Networks, Ph.D. Thesis, Dept. of Electrical and Electronic Engineering, University of Cagliari, 2013, pp. 18-18, Available: http://veprints.unica.it/868/1/Pilloni_PhD_Thesis.pdf

[24] Noufal K.P, Wireless Sensor Networks – Scalability and Performance Issues: A Review, IJCST, 2015, pp. 1-1, Available: http://www.ijcst.com/vol61/1/31-Noufal-K-P.pdf

[25] Holger Karl, Andreas Willig, Protocols and Architectures for Wireless Sensor Networks, Wiley, 2005, pp. 63-73

[26] Muhammad R Ahmed, Xu Huang, Dharmandra Sharma, Hongyan Cui, Wireless, Sensor Networks: Characteristics and Architectures, IJECE, Vol. 6, No. 12, 2012, pp. 1398-1400, Available: http://waset.org/publications/9345/wireless-sensor-network-characteristics-and-architectures

[27] John A. Stankovic, Anthony D. Wood, Tian He, Realistic Applications for Wireless Sensor Networks, Springer, 2010, pp. 835-861, Available: https://link.springer.com/chapter/10.1007%2F978-3-642-14849-1_25#page-1

[28] M. P. Đurišić, Z. Tafa, G. Dimić and V. Milutinović, A survey of military applications of wireless sensor networks, Mediterranean Conference on Embedded Computing (MECO), 2012, pp. 196-199, Available: http://home.etf.rs/~vm/ppt/1%20A_Survey_Military_Apps_WSNs_MPDj.ZT.GD.VM.pdf

[29] Milan Erdelj, Nathalie Mitton, Enrico Natalizio, Applications of Industrial Wireless Sensor Networks, FUN Research Group, INRIA Lille, 2012, pp. 9-20, Available: https://www.utc.fr/~enataliz/dokuwiki/_media/en/crc2012.pdf

[30] S. Idwan, J. A. Zubairi and I. Mahmood, Smart Solutions for Smart Cities: Using Wireless Sensor Network for Smart Dumpster Management, International Conference on Collaboration Technologies and Systems (CTS), Orlando, FL, 2016, pp. 493-497, Available: doi: 10.1109/CTS.2016.0092

[31] M. L. Rajaram, E. Kougianos, S. P. Mohanty and U. Choppali, Wireless sensor network simulation frameworks: A tutorial review: MATLAB/Simulink bests the rest, IEEE Consumer Electronics Magazine, vol. 5, no. 2, 2016, pp. 63-69, Available: doi: 10.1109/MCE.2016.2519051

[32] A. K. Dwivedi1, O. P. Vyas, An Exploratory Study of Experimental Tools for Wireless Sensor Networks, Published Online, July 2011, pp. 215-240, Available: doi:10.4236/wsn.2011.37025

[33] H.M.A. Fahmy, Wireless Sensor Networks: Concepts, Applications, Experimentation and Analysis, Springer, 2016, pp. 248-250

[34] Georgios Z. Papadopoulos, Kosmas Kritsis, Antoine Gallais, Periklis Chatzimisios, and Thomas Noël , Performance Evaluation Methods in Ad Hoc and Wireless Sensor Networks: A Literature Study, IEEE Communications Magazine, vol. 54, no. 1, 2016, pp. 122-128, doi: 10.1109/MCOM.2016.7378437

[35] Fei Yu, A Survey of Wireless Sensor Network Simulation Tools, 2014, Available: http://www1.cse.wustl.edu/~jain/cse567-11/ftp/sensor/index.html, Accessed: 2017-03-27

[36] R. G. Sargent, Verification and validation of simulation models, 2007 Winter Simulation Conference, Washington, DC, 2007, pp. 124-137. doi: 10.1109/WSC.2007.4419595

[37] MSPsim, Available: https://www.sics.se/projects/mspsim-a-java-based-simulator-of-msp430-sensor-network-platforms, Accessed: 2017-03-27

[38] Avrora, Available: http://compilers.cs.ucla.edu/avrora/, Accessed: 2017-04-02

[39] MSP430™, ultra-low-power microcontrollers, Available: www.ti.com/general/docs/marketurl.tsp?name=msp430, Accessed: 2017-04-02

[40] AVR, Atmel AVR microcontroller, Available: http://www.atmel.com/products/microcontrollers/avr/, Accessed: 2017-04-02

[41] TinyOS Documentation Wiki – TinyOS, Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki, Accessed: 2017-04-04

[42] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, Thiemo Voigt, Demo Abstract: Cross-level Simulation in COOJA, IEEE Conference on Local Computer Networks, 2006, pp. 641-648, Available: doi: 10.1109/LCN.2006.322172

[43] Bartosz Musznicki, Piotr Zwierzykowski, Survey of Simulators for Wireless Sensor Networks, International Journal of Grid and Distributed Computing Vol. 5, No. 3, 2012, pp. 23-45, Available: http://www.sersc.org/journals/IJGDC/vol5_no3/3.pdf

[44] AlgoSenSim - Overview [TCS-Sensor Lab], Available: tcs.unige.ch/doku.php/code/algosensim/overview, Accessed: 2017-04-08

[45] Kevin Roussel, Ye-Qiong Song, Olivier Zendra, Using Cooja for WSN Simulations: Some New Uses and Limits. EWSN 2016 - NextMote workshop, Junction Publishing, Proceedings of the 2016 International Conference on Embedded Wireless Systems and Network - EWSN 2016, pp.319-324, Available: https://hal.inria.fr/hal-01240986v2/document

[46] CONET Simulation platform, Available: https://www.cooperating-objects.eu/testbed-simulation/simulation-platform/, Accessed: 2017-04-11

[47] Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, and Sabrina Sicari, A Comparative Study of Recent Wireless Sensor Network Simulators. ACM Trans. Sen. Netw. 12, 3, Article 20, (July 2016), pp. 1-11. Available: DOI: http://dx.doi.org/10.1145/2903144

[48] Eriksson, Joakim and Österlind, Fredrik and Finne, Niclas and Tsiftes, Nicolas and Dunkels, Adam and Voigt, Thiemo and Sauter, Robert and Marrón, Pedro José, COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks, 2nd International Conference on Simulation Tools and Techniques (SIMUTools), 2009, pp. 1-7, Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5637

[49] Carlo Alberto Boano, Kay Romer, Fredrik Osterlind, Thiemo Voigt, Demo Abstract: Realistic Simulation of Radio Interference in COOJA, 2011, pp. 1-2, Available: http://soda.swedish-ict.se/4109/1/boano11realistic-demo.pdf

[50] Pedro José Marron, Stamatis Karnouskos, Daniel Minder, Aníbal Ollero, The Emerging Domain of Cooperating Objects, Springer-Verlag Berlin Heidelberg, Chapter 2, 2011, pp. 13-18

[51] NS-3 Simulator Basics, Available: http://www.iitg.ernet.in/cse/rana2013/material/day5part2/presentation.pdf, Accessed: 2017-04-18

[52] NS-3 Documentation, Available: https://www.nsnam.org/doxygen/index.html, Accessed: 2017-04-20

[53] Hiemo Voigt, Joakim Eriksson, Fredrik Österlind, Robert Sauter, Nils Aschenbruck, Pedro J. Marrón, Vinny Reynolds, Lei Shu, Otto Visser, Anis Koubaa, and Andreas Köpke, Towards comparable simulations of cooperating objects and

wireless sensor networks. In Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '09), 2009, pp.1-10, Available: doi: https://doi.org/10.4108/ICST.VALUETOOLS2009.7651

[54] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. 2003. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03). 2003, pp. 126-137, Available: doi=http://dx.doi.org/10.1145/958491.958506

[55] Laurynas Riliskis, Evgeny Osipov, Symphony: A Framework for Accurate and Holistic WSN Simulation, Sensors 2015, Vol. 15(3), 2015, pp. 4677-4699; Available: doi:10.3390/s150304677

[56] Jingyao Zhang, Yi Tang, Sachin Hirve, Srikrishna Iyer, Patrick Schaumont, and Yaling Yang, A Software-Hardware Emulator for Sensor Networks, 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, , 2011, pp. 440-448, Available: doi: 10.1109/SAHCN.2011.5984928

[57] OMNeT++ Documentation, Available: https://omnetpp.org/documentation, Accessed: 2017-04-23

[58] MiXiM, Available: https://omnetpp.org/9-articles/software/3642-mixim10released, Accessed: 2017-04-23

[59] András Varga, The OMNET++ Discrete Event Simulation, System, Proceedings of the European Simulation Multiconference (ESM'2001), 2001, pp. 1-7, Available: https://omnetpp.org/download/docs/papers/esm2001-meth48.pdf

[60] SENSE, Available: http://www.ita.cs.rpi.edu/, Accessed: 2017-04-24

[61] Viptos, Available: http://ptolemy.eecs.berkeley.edu/viptos/viptos1.0.htm, Accessed: 2017-04-24

[62] Shafiullah Khan, Al-Sakib Khan Pathan, Nabil Ali Alrajeh, Wireless Sensor Networks: Current Status and Future Trends, Accessed: 2017-04-24

[63] Antoine Fraboulet, Guillaume Chelius, Eric Fleury. Worldsens: development and prototyping tools for application specific wireless sensors networks. ACM. 6th ACM/IEEE international conference on Information Processing in Sensor Networks (IPSN 2007), 2007, pp. 176-185, Available: https://hal.inria.fr/inria-00384835/file/ipsn-07.pdf

[64] WSNet, Avaiable: http://wsnet.gforge.inria.fr/index.html, Accessed: 2017-04-24

[65] WSim, Available: http://wsim.gforge.inria.fr/index.html, Accessed: 2017-04-24

[66] MICA, Available: http://www.memsic.com/wireless-sensor-networks/, Accessed: 2017-04-25

[67] Sky mote data sheet, Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf, Accessed: 2017-04-30

[68] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne and T. Voigt, Cross-Level Sensor Network Simulation with COOJA, 31st IEEE Conference on Local Computer Networks, Tampa, FL, 2006, pp. 641-648, Available: doi:10.1109/LCN.2006.322172

[69] Fredrik Österlind, A Sensor Network Simulator for the Contiki OS, Master Thesis report, SICS, 2006, pp. 1-40, Available: https://www.diva-portal.org/smash/get/diva2:1041560/FULLTEXT01.pdf

[70] A. Dunkels, B. Gronvall and T. Voigt, Contiki - a lightweight and flexible operating system for tiny networked sensors, 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455-462
Available: doi: 10.1109/LCN.2004.38

[71] VMware, Available: https://www.vmware.com/, Accessed: 2017-05-06